НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«До захисту допущено»
В.О. Завідувача кафедри
_____ О.Л. Тимощук

Дипломна робота

на здобуття ступеня бакалавра
з напряму підготовки 6.040303 Системний аналіз
на тему: «Покращення прогнозування часових рядів за допомогою підходу з використанням bootstrap aggregating»

Виконав:
студент IV курсу, групи КА-61
Островерхий Антон Сергійович _____

Керівник:
доцент кафедри ММСА, к.ф-м.н.
Яковлева А.П. _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент кафедри ММСА, к.т.н.
Коваленко А.Є._____

Рецензент:
доцент кафедри ММСА, к.ф-м.н.
Ільєнко А.Б._____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших
авторів без відповідних посилань.
Студент _____

Київ – 2020

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Островерхому Антону Сергійовичу

1. Тема роботи «Покращення прогнозування часових рядів за допомогою підходу з використанням bootstrap aggregating», керівник роботи доцент кафедри ММСА, к.ф-м.н. Яковлева А.П, затверджені наказом по університету від «_25_»_травня_ 20_20_ р. №_1143-с__

2. Термін подання студентом роботи 08 травня 2020 року

3. Вихідні дані до роботи

1. Мова програмування Python 3
2. Середовище розробки – Jupyter Notebook
3. Бібліотеки, що використовувалися: Pandas,SciKit, Seaborn, Matplotlib, NumPy

4. Зміст роботи

1. Проаналізувати фінансовий часовий ряд

2. Розробити програмно три моделі різної природи для прогнозування фінансового часового ряду
3. Розробити модель з використанням bootstrap aggregating та порівняти результати роботи усіх моделей
4. Виконати економічний аналіз програмного продукту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Презентація

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	к.е.н., доц. Шевчук О. А.	21.04.20	31.05.20

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	13.04.20	
2	Аналіз існуючих моделей	22.04.20	
3	Підбір показників для оцінки моделі	28.04.20	
4	Підбір вхідних даних для моделі	5.05.20	
5	Розробка продукту для моделювання та системи оцінювання моделі	12.05.20	
6	Тестування продукту, отримання результатів	15.05.20	
7	Оформлення дипломної роботи	18.05.20	

Студент: Островерхий А.С.

Керівник: Яковлева А.П

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Дипломна робота: 64 с., 30 рис., 6 табл., 2 дод., 7 джерел.

ПОКРАЩЕННЯ ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ ЗА ДОПОМОГОЮ ПІДХОДУ З ВИКОРИСТАННЯМ BOOTSTRAP AGGREGATION.

Метою даної роботи: дослідження можливості та властивостей застосування моделей bagging, Random Forest та AdaBoost для прогнозу фінансового часового ряду в умовах корельованих даних та порівняння помилки роботи кожної з моделей.

Актуальність дослідження: прогнозування фінансового часового ряду компанії Yahoo Finance за великий проміжок часу (більше 20 років) з можливістю включення найактуальніших на сьогодні даних у ряд. Моделі тренуються на даних, котрі містять інформацію про економічну кризу 2007-2008 років, що робить їх стійкими в умовах можливої кризи після пандемії коронавірусу.

Об'єкт дослідження: набір фінансових даних компанії Yahoo! Finance з 1998 року.

Предмет дослідження: моделі Bagging, Random Forest та AdaBoost для прогнозування часового ряду, засновані на бустрепінгу та бустингу, їх особливості та застосування на реальному фінансовому часовому ряді.

Методи дослідження: застосовані моделі прогнозування різної природи для часового фінансового ряду, виконані на мові програмування Python у середовищі Jupyter Notebook з використанням пакету sklearn.

Отримані результати: Побудовано модель бустингу AdaBoost з непоганим значенням помилки, котра не перенавчилась на вибірці. Досягнено кращого за Random Forest результату, котрий вважається оптимальним алгоритмом для ряду такої природи як у Yahoo! Finance.

ABSTRACT

Thesis on 65 pages, contains 30 figures, 6 tables, 2 appendices, 7 sources.

IMPROVING TIME FORECASTING USING THE BOOTSTRAP AGGREGATION APPROACH

The purpose of this work: to study the feasibility and properties of using models of bagging, Random Forest and AdaBoost to forecast the financial time series in the context of correlated data and compare the error of each model.

Relevance of the study: forecasting the financial time series of Yahoo Finance for a long period of time (more than 20 years) with the possibility of including the most relevant data in the series. The models are based on data that contain information about the economic crisis of 2007-2008, which makes them resilient in the event of a possible crisis after a coronavirus pandemic.

Object of study: a set of financial data from Yahoo! Finance since 1998. Subject of research: Bagging, Random Forest and AdaBoost models for time series forecasting, based on boosting and boosting, their features and application on a real financial time series.

Research methods: applied forecasting models of different nature for the financial time series, made in the Python programming language in the Jupyter Notebook environment using the sklearn package.

Results obtained: The AdaBoost boosting model was built with a good error value, which was not retrained in the sample.

Achieved a better result than Random Forest, which is considered the optimal algorithm for a number of such nature as Yahoo! Finance. The best result was achieved with the bootstrap aggregating approach

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

RF – алгоритм Random Forest

МО – машинне навчання

np – бібліотека NumPy

pd – бібліотека Pandas

DS – датасайенс

JN – середовище Jupyter Notebook

ВСТУП

Дослідження часових рядів з метою їх подальшого прогнозування – це задача, що зустрічається у будь якій галузі. Будь-то промисловість, де часовий ряд буде характеризувати кількість залишкової сировини за годину, день чи тиждень. Господарство, де важливо розуміти розміри майбутнього врожаю також зустрінеться з часовими рядами, дослідивши які буде можливо побудувати модель, враховуючи усі попередні нюанси врожайності та зробити її стійкою до змін та нових подій, котрі не зустрілися за попередні роки. Продажі, магазини, погода тощо. Аналіз та прогнозування часових рядів зустрінеться всюди. Тому, якісне її розв'язання, що представляє собою якісну модель – задача, котру слід розв'язати з максимально точно, наскільки дозволяє конкретна ситуація.

Якщо вважати обчислювальні можливості скінченним, що, безумовно, так і є, та зважати, що часові ряди можуть мати різні розміри, наприклад, данні про врожайність за 10 років на території України та данні по продажу кави у кав'ярні за минулий квартал – це зовсім не однаковий об'єм даних.

В першому випадку можна побудувати більш точну модель, тому що даних велика кількість. Але обмеження буде на обчислювальні можливості, адже більш точна модель потребуватиме більше ресурсів в додаток до більшої кількості даних.

В другому випадку, побудова складної моделі може не матиме багато користі, адже на невеликому обсязі даних, точність складної моделі може не набагато відрізнятися від простої, а час на побудову більш насиченої непропорціональний її користі. Тому, доцільно використати просту, але дієву модель.

Не тільки розміри початкової вибірки впливає на вибір алгоритму, а й те, які властивості мають елементи цієї вибірки. Маючи навіть трохи знань

про це уже можливо вибрати початкові моделі. Коли це зроблено, слід намагатися покращити результати. І тут доцільно спробувати bootstrap aggregating, ця методика проста у розумінні, а приріст точності при мінімальних затратах у більшості задач значний.

Дипломна робота спрямована на доведення ефективності bagging (інша назва Bootstrap aggregating) при покращені точності прогнозу часового ряду та порівняння з іншими популярними методами.

Часовий ряд, що буде використано для прикладу – це реальні дані про акції компанії Yahoo! Finance за останні 20 років, що можна знайти у відкритому доступі.

На практиці, застосування бегінгу на вже готових моделях, покращить точність при мінімальних затратах ресурсів у більшості задач, навіть на датасеті за 20 років.

В дипломній роботі, окрім бегінгу, будуть розглянуті й інші моделі, а саме AdaBoost – ефективний бустинг та Random Forest – алгоритм, котрий себе найкраще проявляє на схожих до робочої вибірки датасетах.

Предметом дослідження є поведінка алгоритму bootstrap aggregation для прогнозу часового ряду за даними акцій компанії Yahoo за останні 20 років.

Датасет включає у себе дані з 1998 року, у ньому є дані для економічної кризи 2007-2008 років. Ці дані є безцінними в межах даної дипломної роботи. Оскільки коронавірус ставить економіку у скрутне становище, дуже ймовірно що слідує глобальна економічна криза.

Наскільки важливе прогнозування акцій у період кризи? Наскільки це можливо. Саме дані за 2007-2008 роки кризи у поєднанні з періодами росту та стабільності дозволять побудувати моделі прогнозу фінансового часового ряду, акутальні на сьогодні.

1 АЛГОРИТМИ

1.1 Опис Random Forest

Випадковий ліс представляє собою універсальний алгоритм, котрий використовується в сучасній практиці для вирішення задач класифікації, регресії та кластеризації. У кожному випадку діють багато дерев прийняття рішень, а результати їх роботи представляються у різному вигляді для кожної із задач. Так, для класифікації, фінальне рішення приймається за більшою кількістю голосів, де кожен голосуючий – одне з дерев, а голос – відповідно клас, до якого було віднесено елемент. Схему роботи випадкового лісу можна побачити на наступному рисунку (рисунок 1.1).

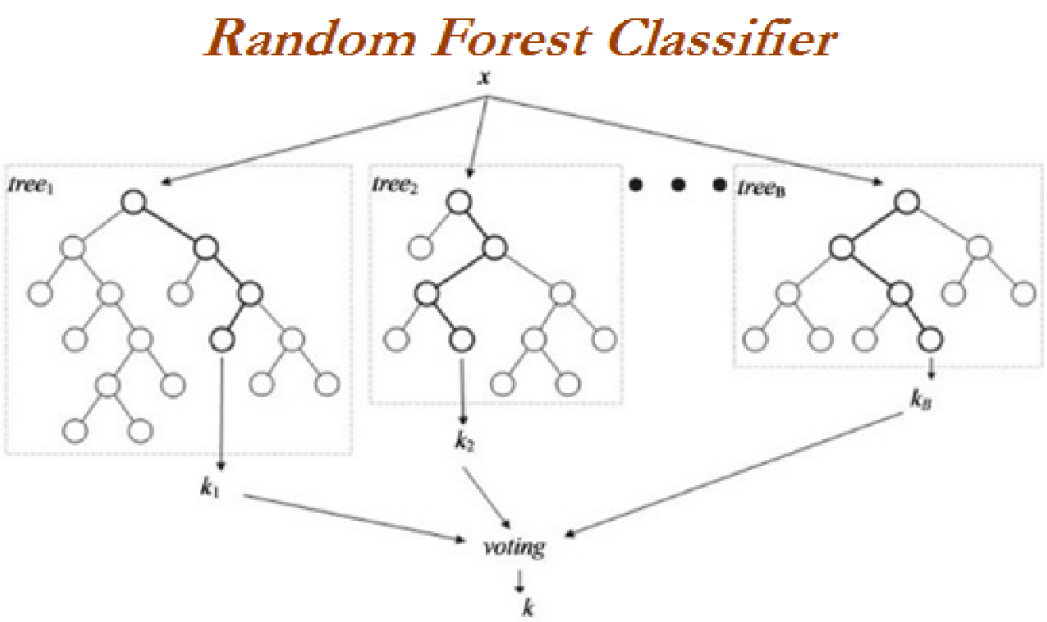


Рисунок 1.1 – Принцип роботи Random Forest

Для задач регресії та прогнозу часових рядів, кожне дерево приймає рішення, але остаточний прогноз – це усереднення кожного індивідуального рішення.

Хоча Random Forest справляється з більшістю задач прогнозу, на вибірці, що використовується у дипломній роботі, є певні аутлаєри. Вони суттєво впливають на кожне індивідуальне рішення дерева, що, в свою чергу, менш суттєво, але вплине на остаточне рішення всієї моделі. Те, як реалізація bagging подолає цей недолік випадкового лісу буде наведено у наступних розділах.

1.2 Опис AdaBoost

Адаптивний бустинг - це мета алгоритм машинного навчання, котрий об’єднує базові алгоритми у композицію в процесі навчання, з метою покращення остаточного прогнозу. Кожний етап роботи можна назвати раундом. Широко розповсюджений на практиці адаптивний бустинг AdaBoost.

Так, на основі, наприклад, п’яти раундів виглядатиме принцип роботи AdaBoost.

Будуватиметься ансамбль класифікаторів.

Перший: перевіряється точність роботи кожного класифікатора на частині даних з training датасету та обирається найкращих з них. Він додається до ансамблю, а для тих елементів, котрі були помилково віднесені до іншого класу збільшується вага. Таким чином, наступні класифікатори більш ймовірно звернуть на них увагу.

Другий: повторюються дії першого раунда. Відмінність у тому, що на цьому етапі неправильно класифіковані елементи мають більшу вагу, тому частіше потраплятимуть до нової вибірки. Кращий класифікатор у цьому раунді також додається до ансамблю, а його помилки обробляються так само, як у першому раунді.

Третій – Четвертий раунд: аналогічно першим двум.

П’ятий: остаточно сформовано ансамбль класифікаторів.

Формально, роботу АдаБуст можно записати так:

Дано: X^l - обучаюча вибірка;

$b_1, \dots, b_T$ - базові алгоритми класифікації;

1. Ініціалізуємо ваги елементів:

$$p_i = \frac{1}{l}, \text{де } i = 1, \dots, l; \qquad (1.1)$$

2. Для будь-якого t = 1,…, T, доки не виконується критерій зупинки.

2.1 Знаходимо класифікатор

$$b_t: X \to \{-1, +1\}, \text{де } b_t = \arg\min_l Q(b, W^l); \qquad (1.2)$$

2.2 Змінюємо коефієнт взваженого голосування для b_t:

$$\alpha_t = \frac{1}{2} \ln \frac{1-Q(b,W^l)}{Q(b,W^l)}, \text{де } Q(b, W^l) \text{ з (1.2)}; \qquad (1.3)$$

2.3 Обчислюємо вагу елементів заново:

$$\omega_i = \omega_i \exp\big(-\alpha_t y_i b_t(x_i)\big), i = 1, \dots, l; \qquad (1.4)$$

2.4 Нормуємо іх:

$$\omega_0 = \sum_{i=1}^{l} \omega_j; \text{ де } \omega_j = \frac{\omega_i}{\omega_0}, i = 1,\dots,l; \qquad (1.5)$$

3. Остаточно повертаємо:

$$a(x) = sign\left(\sum_{i=1}^{T} \alpha_i \, b_i(x)\right), \text{де } b_i \text{ з формули (1.2)}; \qquad (1.6)$$

AdaBoost застосовується на практиці, оскільки добре узагальнює, на відмінну від Random Forest. На вибірці даних про акції Yahoo Finance повинен добре себе проявити. Також, цей метод не потребує великих обчислювальних ресурсів. І, нарешті, те чого у даному датасеті багато – шумові виброси. Адаптивний бустинг добре їх ідентифікує, знову, у противагу випадковому лісу.

1.3 Опис Bootstrap Aggregation

Інакша назва бегінг. Являє собою алгоритм машинного навчання із класу ансамблей. Складається з двох частин: агрегації та завантаження. Ансамблеві техніки навчання – це підгрупа мульти-класифікаторного навчання.

Головна ідея – використання декількох моделей з одним й тим самим алгоритмом навчання з метою покращення точності прогнозу.

Bootstrapping (завантаження), в свою чергу, метод розділення датасету на семпли – вибірки меншого розміру, до яких і будуть застосовані алгоритми. Семпли на цьому етапі повертаютсья назад у загальний датасет, для того, щоб натсупниі були обрані так само випадково і не залежали від попередніх.

Aggregation (агрегація), у цьому випадку, представляє собою використання окремого алгоритма до кожної випадкової підвибірки з етапу завантаження. В результаті роботи яких будуть побудовані окремі класифікатори зі своїми прогнозами, на основі яких побудується комбінований класифікатор, прогноз якого і буде остаточним.

Загальний принцип роботи бегінгу можна побачити на наступному рисунку (1.2).

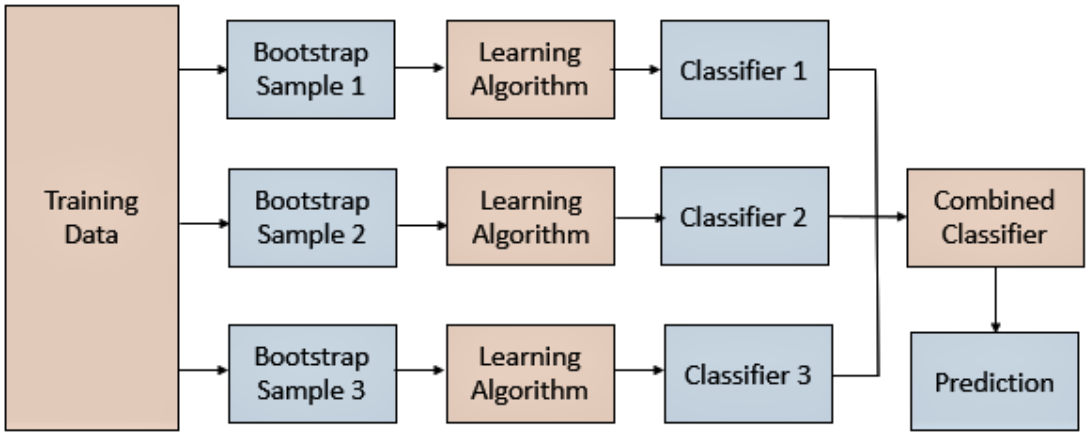


Рисунок 1.2 – Принцип роботи Bootstrap Aggregation

Головним конкурентом бегінгу в межах даної дипломної роботи є AdaBoost, а у загальній практиці – алгоритми бустингу.

Сильних сторін цього підходу багато, наприклад, простота імплементації та реалізації, висока точність тощо.

Основними недоліками є втрата інтерпретованості моделі та чутливість до перенавчання. Незважаючи на високу точність, неправильне виділення підвиборок (семплів) може повністю зруйнувати роботу алгоритму.

На датасеті, що використовується у дипломній роботі, використання бегінгу стане ресурсозатратною процедурою, на відміну від бустингу. Але покращення точності прогнозу нівелює різницю в використанні обчислювальних ресурсів.

2 ВИБІРКА

2.1 Опис

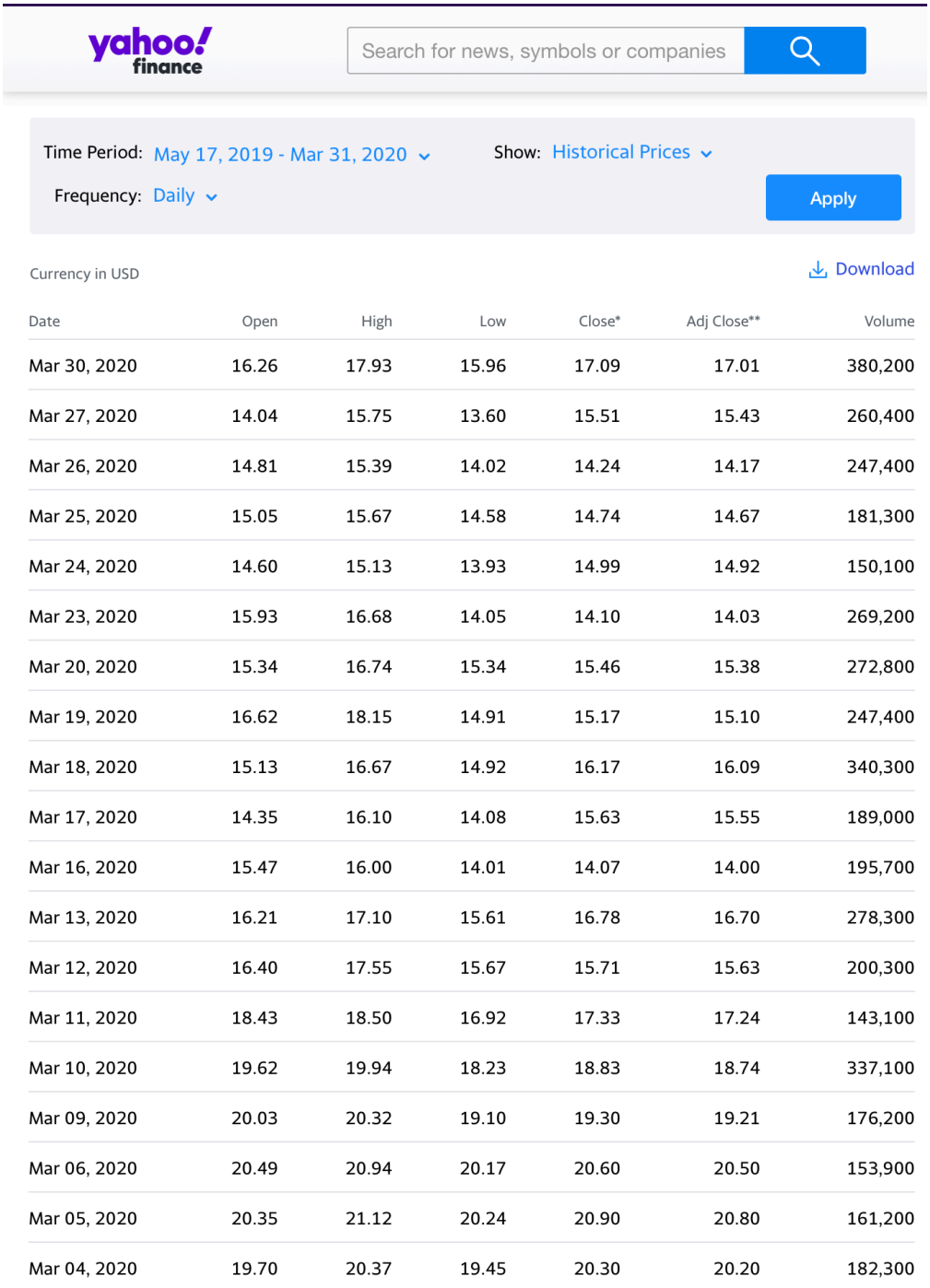
Yahoo! Finance представляє інформацію о фінансах компанії Yahoo! з 1998 року. Являє собою головного поставника такої інформації у США зі стамільйоною аудиторією станом на 2020 рік. Дані представляють собою інформацію про рейтинги акцій та фінансових звітів компанії.

Компанія підтримує використання датасету у навчальних цілях та дає можливість користуватися їм у повному обсязі. Також, постачає його у різних форматах для використання у навчальних проектах та досліджень області Data Science. Хоча є можливість застосувати цю вибірку у більш зручному форматі, для дипломної було обрано користуватися датасетом в режимі реального часу через сервіс, котрий компанія надала для мови програмування Python.

Дані для аналізу взято за 10 років: з 1 січня 2010 до 31 березня 2020 року. Такий об’єм даних повністю забезпечить різноманітним часовим рядом будь яку модель, в тому числі й AdaBoost, Random Forest та Bootstrap Aggregation в межах цієї роботи.

Було обрано саме цю вибірку, оскільки вона динамічно змінюється з часом та є можливість оновлювати моделі кожен день за необхідністю постійно додаючи нові об’єкти. Також, дані повністю покривають усі слабкі моменти кожної моделі, які були описані у попередніх пунктах, що ставить кожну з них у справедливе порівняння з іншими.

На наступному рисунку (2.1) наведено початковий вигляд вибірки з офіційного сайту. Для подальшої роботи над датасетом, його було оброблено чином, вказаним далі в цьому пункті.



Date	Open	High	Low	Close*	Adj Close**	Volume
Mar 30, 2020	16.26	17.93	15.96	17.09	17.01	380,200
Mar 27, 2020	14.04	15.75	13.60	15.51	15.43	260,400
Mar 26, 2020	14.81	15.39	14.02	14.24	14.17	247,400
Mar 25, 2020	15.05	15.67	14.58	14.74	14.67	181,300
Mar 24, 2020	14.60	15.13	13.93	14.99	14.92	150,100
Mar 23, 2020	15.93	16.68	14.05	14.10	14.03	269,200
Mar 20, 2020	15.34	16.74	15.34	15.46	15.38	272,800
Mar 19, 2020	16.62	18.15	14.91	15.17	15.10	247,400
Mar 18, 2020	15.13	16.67	14.92	16.17	16.09	340,300
Mar 17, 2020	14.35	16.10	14.08	15.63	15.55	189,000
Mar 16, 2020	15.47	16.00	14.01	14.07	14.00	195,700
Mar 13, 2020	16.21	17.10	15.61	16.78	16.70	278,300
Mar 12, 2020	16.40	17.55	15.67	15.71	15.63	200,300
Mar 11, 2020	18.43	18.50	16.92	17.33	17.24	143,100
Mar 10, 2020	19.62	19.94	18.23	18.83	18.74	337,100
Mar 09, 2020	20.03	20.32	19.10	19.30	19.21	176,200
Mar 06, 2020	20.49	20.94	20.17	20.60	20.50	153,900
Mar 05, 2020	20.35	21.12	20.24	20.90	20.80	161,200
Mar 04, 2020	19.70	20.37	19.45	20.30	20.20	182,300

Рисунок 2.1 – Вигляд вибірки у первинному вигляді на офіційному сайті Yahoo! Finance

Так виглядає частина вибірки (рисунок 2.2), завантажена у режимі реального часу, за допомогою вбудованого сервісу для Python, що використовується у моделях.

Date	Volume	Today	Lag1	Lag2	Lag3
2000-01-07	10505400	6.101049	-6.003584	-14.874142	-8.321678
2000-01-10	14757900	-0.539084	6.101049	-6.003584	-14.874142
2000-01-11	10532700	-3.523035	-0.539084	6.101049	-6.003584
2000-01-12	10804500	-4.775281	-3.523035	-0.539084	6.101049
2000-01-13	10448100	3.736480	-4.775281	-3.523035	-0.539084
...	...	...	...	...	...
2020-03-25	6479100	-2.796764	1.958663	3.073522	-1.852280
2020-03-26	6221300	3.693316	-2.796764	1.958663	3.073522
2020-03-27	5387900	-2.832539	3.693316	-2.796764	1.958663
2020-03-30	6126100	3.360348	-2.832539	3.693316	-2.796764
2020-03-31	5123600	-0.724559	3.360348	-2.832539	3.693316

5089 rows × 5 columns

Рисунок 2.2 – Вигляд робочої вибірки

Опис полів:

Date – точна дата, коли були зафіксовані значення

Volume – об’єм прибутку компанії за цей день

Today – повернення вкладень компанії за цей день

Lag1 - повернення вкладень компанії за попередній день

Lag2 - повернення вкладень компанії за два дні до дати у стовпчику Date

Lag3 - повернення вкладень компанії за три дні до дати у стовпчику Date

Таким чином, за вибіркою утворена матриця 5089 (кількість днів) на 5 (кількість характеристик), тобто з 25445 об’єктів.

2.2 Доцільність вибірки

Дані компанії містять у собі інформацію у період економічної кризи 2007-2008 років з великою кількістю стрибків, непередбачуваного спаду у 2006 році (іпотечний кризис Америки) та дані у період виходу із кризи з подальшою стабільністю та ростом. Такий датасет є безцінним та найактуальнішим на сьогодні. Оскільки у ньому є фінансові дані у період економічної кризи, що дозволяє натренувати моделі, котрі будуть ефективними для прогнозування фінансових часових рядів у період можливої економічної кризи після завершення пандемії коронавірусу.

Вищевказані три алгоритми будуть застосовані для прогнозування щоденних прибутків для запасів компанії, використовуючи попередні дані за три дні повернення вкладень.

Це складне завдання не в останню чергу через те, що дані, мають низьке співвідношення сигнал / шум, а й тому, що такі дані послідовно корельовані. Це означає, що обрані вибірки не є справді незалежними одна від одної, що може мати сумні наслідки для статистичної обґрунтованості процедури.

Наразі буде застосовано стандартний розрив навчальних тестувань, оскільки в дипломній роботі акцент робиться на порівнянні помилок між моделями, а не на абсолютній помилці, досягнутій для кожної.

У наступних розділах буде проведено надійну процедуру за допомогою механізму перехресної перевірки часових рядів Scikit-Learn.

Кінцевим результатом буде графік середньої квадратичної помилки (MSE) кожного метода (бегінг, випадковий ліс та адаптивний бустинг) від кількості оцінок, використаних у вибірці.

Дані у цьому датасеті дають змогу повністю розкритися кожному з алгоритмів, явно виявляючи слабкі та сильні сторони кожного.

Так, для Random Forest буде важко справлятися з впливом аутлаєрів, AdaBoost зіткнеться з можливістю перевнавчання при зростанні кількості естіматорів, а bagging з обчислювальними складностями та особливістю самих даних.

3 МОДЕЛІ

3.1 Визначення

У дипломній роботі будуть використані три моделі (Random Forest, AdaBoost, Bootstrap Aggregating) та усі вони – представники ансамблевих методів машинного навчання.

Ансамблевий метод - це метод, який поєднує в собі безліч слабких учнів, заснованих на одному і тому ж алгоритмі навчання, з метою створення (сильнішого) учня, ефективність якого краща, ніж будь-якого з окремих. Ансамблеві методи допомагають зменшити зміщення та / або дисперсію.

Оскільки у методів машинного навчання є глобальна проблема, як помилка дисперсії, викликана чутливістю до невеликих змін у навчальному наборі. Коли дисперсія велика, це означає, що алгоритм буде знову узгоджений з навчальним датасетом, і тому навіть мінімальні зміни навчального датасету можуть давати кардинально різні прогнози. Замість моделювання узагальнення у трейнінг підвибірці алгоритм помилково приймає шум для сигналу.

Тому, представники ансамблевих методів машинного навчання мають свої переваги і дуже добре підходять для конкретної задачі.

Окрім проблеми з дисперсію, можуть виникати і наступні труднощі.

1) Зсув: ця помилка викликана нереальними припущеннями. Коли зсув великий, це означає, що алгоритм машинного навчання не зміг розпізнати важливі зв’язки між ознаками та результатами.
2) Шум: ця помилка викликана дисперсією спостережуваних значень, таких як непередбачувані зміни або помилки вимірювання. Це фатальна помилка, яку не може подолати жодна з моделей.

Для реалізації моделей використано мову програмування Python.

Python являється інтерперетуємой об'єктно-орієнтованой мовою програмування високого рівня з динамічною семантикою. Його вбудовані структури даних високого рівня в поєднанні з динамічною типізацією і динамічним зв'язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання в якості скриптового або сполучного мови для з'єднання існуючих компонентів. Простий, легкий в освоєнні синтаксис Python підкреслює легкість для читання і, отже, знижує вартість обслуговування програми. Python підтримує модулі та пакети, що сприяє модульності програми і повторного використання коду. Інтерпретатор Python і велика стандартна бібліотека доступні в вихідному або довічним вигляді безкоштовно для всіх основних платформ і можуть вільно поширюватися.

Часто програмісти використовують в Python через підвищену продуктивность, яку він забезпечує. Оскільки етап компіляції відсутній, цикл edit-test-debug неймовірно швидкий. Налагодження програм на Python просте: помилка або неправильне введення ніколи не викличуть помилку сегментації. Замість цього, коли інтерпретатор виявляє помилку, він викликає виключення. Коли програма не перехоплює виняток, інтерпретатор друкує трасування стека.

Відладчик на рівні вихідного коду дозволяє перевіряти локальні і глобальні змінні, оцінювати довільні вирази, встановлювати точки зупинки, крок за кроком проходити через код тощо. Відладчик написаний на самому Python, що свідчить про його інтроспективну силу. З іншого боку, часто найшвидший спосіб налагодження програми - це додати кілька операторів print до джерела: швидкий цикл edit-test-debug робить цей простий підхід дуже ефективним.

3.2 Використані бібліотеки

Цей розділ корелює з наступним, у якому наведено реалізацію моделей. Варто звернути уваги наскільки читаємо та негроміздко виглядає реалізація моделей та прогромний код вцілому. Цього не вдалось би досягти без підключення пакетів із цього розділу.

Scikit-learn надає ряд алгоритмів навчання з вчителем та без через узгоджений інтерфейс в Python.

Він ліцензується за роздільною спрощеною ліцензією BSD і поширюється в багатьох дистрибутивах Linux, заохочуючи академічне і комерційне використання.

Бібліотека побудована на SciPy (Scientific Python), який повинен бути встановлений перед використанням scikit-learn. Цей стек, який включає в себе:

1)NumPy: базовий пакет n-мірного масиву

2)SciPy: фундаментальна бібліотека для наукових обчислень

3)Matplotlib: Комплексна 2D / 3D графіка

4)IPython: поліпшена інтерактивна консоль

5)Sympy: Символічна математика

6)Pandas: структури даних і аналіз

Розширення або модулі для SciPy зазвичай називаються SciKits. Таким чином, модуль надає алгоритми навчання і називається scikit-learn.

Бачення бібліотеки - це рівень надійності та підтримки, необхідний для використання у виробничих системах. Це означає глибоку увагу до таких проблем, як простота використання, якість коду, спільна робота, документування і продуктивність.

Хоча інтерфейсом є Python, c-бібліотеки дозволяють підвищити продуктивність, наприклад, numpy для масивів і операцій з матрицями, LAPACK, LibSVM і дбайливе використання cython.

Бібліотека орієнтована на моделювання даних. Він не орієнтований на завантаження, маніпулювання і підсумовування даних. Для цих функцій, звернемося до NumPy і Pandas, які будуть описані далі.

Деякі популярні групи моделей, що надаються Scikit-Learn, включають в себе наступні.

Кластеризація (рисунок 3.1): для угруповання нерозмічених даних, таких як KMeans.

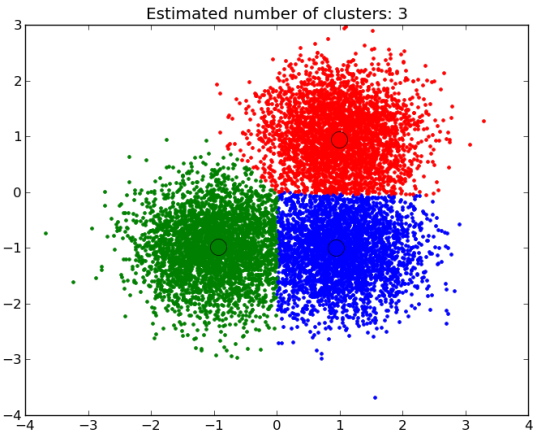


Рисунок 3.1 Візуалізація кластеризації у scikit

Перехресна перевірка: для оцінки продуктивності контрольованих моделей на нечітких даних.

Датасети: для тестових наборів даних і для створення наборів даних з певними властивостями для дослідження поведінки моделі.

Зменшення розмірності: для зменшення кількості атрибутів в даних для підсумовування, візуалізації і вибору функцій, таких як аналіз головних компонентів.

Методи ансамблю: для об'єднання прогнозів кількох контрольованих моделей.

Витяг функції: для визначення атрибутів в графічних і текстових даних.

Вибір характеристик: для визначення значущих атрибутів, з яких можна створювати контрольовані моделі.

Налаштування параметрів: для отримання максимальної віддачі від контрольованих моделей.

Навчання колектора: для узагальнення і відображення складних багатовимірних даних.

Контрольовані моделі: великий масив, не обмежений узагальненими лінійними моделями, дискримінаційним аналізом, наївними байесовськими алгоритмами, ледачими методами, нейронними мережами, машинами опорних векторів і деревами рішень

Наступна необхідна бібліотека без якої втрачається швидкість роботи програми і яка заслуговує того щоб її окремо описати є NumPy. NumPy - це фундаментальний пакет для наукових обчислень на Python. Це бібліотека Python, яка надає об'єкт багатовимірного масиву, різні похідні об'єкти (такі як замасковані масиви і матриці) і набір підпрограм для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції з формами, сортування, вибір, введення / висновок, дискретні перетворення Фур'є , базова лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого.

В основі пакета NumPy лежить об'єкт ndarray. Це інкапсулює n-мірні масиви однорідних типів даних з багатьма операціями, що здійснюються в скомпільованому коді для підвищення продуктивності. Існує кілька важливих відмінностей між масивами NumPy і стандартними послідовностями Python:

Масиви NumPy мають фіксований розмір при створенні, на відміну від списків Python (які можуть динамічно зростати). Зміна розміру ndarray створить новий масив і видалить оригінал.

Всі елементи в масиві NumPy повинні бути одного типу даних і, отже, мати однаковий розмір в пам'яті. Виняток: можна мати масиви об'єктів (Python, включаючи NumPy), що дозволяє використовувати масиви елементів різного розміру.

Масиви NumPy полегшують складні математичні і інші типи операцій з великою кількістю даних.

Як правило, такі операції виконуються більш ефективно і з меншою кількістю коду, ніж це можливо при використанні вбудованих послідовностей Python.

Зростаюча кількість наукових і математичних пакетів на основі Python використовують масиви NumPy; хоча вони зазвичай підтримують введення послідовності Python, вони перетворять такий спосіб введення в масиви перед обробкою і часто виводять масиви NumPy. Іншими словами, щоб ефективно використовувати більшу частину сучасного наукового / математичного програмного забезпечення на основі Python, недостатньо просто знати, як використовувати вбудовані типи послідовностей Python - потрібно також знати, як використовувати масиви NumPy.

Як саме це прискорює роботу програми?

Векторизация описує відсутність будь-якого явного зациклення, індексації тощо. У коді - ці речі відбуваються, звичайно, просто «за лаштунками» в оптимізованому, попередньо скомпільованому C-коді. Векторизованний код має багато переваг, серед яких:

векторизованних код більш лаконічний і зручний для читання

менше рядків коду зазвичай означає менше помилок

код більш схожий на стандартні математичні позначення (що полегшує, як правило, правильне кодування математичних конструкцій)

Векторизац ія призводить до більшої кількості «Pythonic» коду. Без векторизації наш код був би завалений неефективними і важкими для читання циклами.

Трансляція - це термін, який використовується для опису неявного поетапного поведінки операцій;

Взагалі кажучи, в NumPy всі операції, не тільки арифметичні операції, а й логічні, побітові, функціональні тощо, Поводяться неявно поелементно, тобто транслюються. Більш того, в наведеному вище прикладі a і b можуть бути багатовимірними масивами однакової форми, або скаляром і масивом, або навіть двома масивами різної форми, за умови, що менший масив «розширюється» до форми більшого розміру. таким чином, що отримана трансляція є однозначною.

Наочний приклад (рисунок 3.2) як це полегшує (рисунок 3.3) написання коду:

```
for (i = 0; i < rows; i++): {
  for (j = 0; j < columns; j++): {
    c[i][j] = a[i][j]*b[i][j];
  }
}
```

Рисунок. 3.2 Перемноження матриць без NumPy

```
c = a * b
```

Рисунок. 3.3 Перемноження матриць з NumPy

Остання з найважливіших бібліотек, що використовується у дипломній роботі є Pandas DataFrame.

Pandas DataFrame - це двовимірна змінна за розміром, потенційно неоднорідна структура табличних даних з позначеними осями (рядки і стовпці). Фрейм даних - це двовимірна структура даних, тобто дані вирівняні в табличній формі по рядках і стовпцях. Pandas DataFrame (рисунок 3.3) складається з трьох основних компонентів: даних, рядків і стовпців.

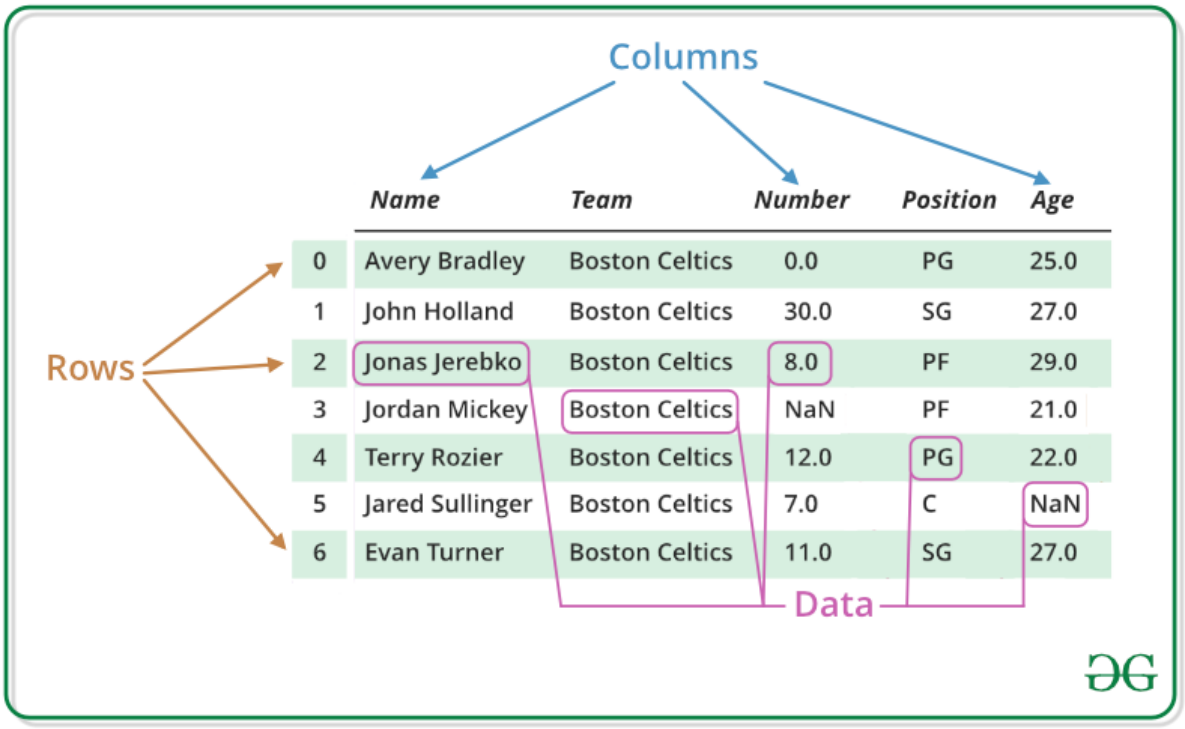


Рисунок. 3.3 Вигляд датафрейму у Pandas

Представлення даних у такому вигляду значно полегшує написання коду та дозволяє з легкістю передавати необхідні структури у наступні естіматори.

Повний список (рисунок 3.4) використаних бібліотек виглядає так.

```
import datetime

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import seaborn as sns
import sklearn
from sklearn.ensemble import (
    BaggingRegressor, RandomForestRegressor, AdaBoostRegressor
)
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.tree import DecisionTreeRegressor
```

Рисунок 3.4 Список залучених бібліотек

Окрім описаних вище, можна побачити такі пакети як seaborn та matplotlib. Обидва пакети використовуються для побудови наглядних графіків, що описують роботу програми та результати прогнозування часового ряду.

Seaborn - це бібліотека для створення статистичної графіки на Python. Він побудований на основі matplotlib і тісно інтегрований зі структурами даних pandas. Ось деякі з функцій, які пропонує Seaborn: API, орієнтований на набір даних, для вивчення відносин між декількома змінними Спеціалізована підтримка використання категоріальних змінних для відображення спостережень або агрегированной статистики Варіанти візуалізації одновимірних або двовимірних розподілів і порівняння їх між підмножинами даних Автоматична оцінка і побудова моделей лінійної регресії для різних видів залежних змінних Зручні види на загальну структуру складних наборів даних Високорівневі абстракції для структурування многосюжетность сіток, які дозволяють легко створювати складні візуалізації Короткий контроль над стилем фігур Matplotlib з декількома вбудованими темами Інструменти для вибору кольорової палітри, які точно відображають шаблони в ваших даних Seaborn прагне зробити візуалізацію центральною частиною вивчення і розуміння даних . Його орієнтовані на набори даних функції побудови графіків працюють з фреймами даних і масивами, що містять цілі набори даних, і внутрішньо виконують необхідне семантичне відображення і статистичне агрегування для створення інформативних графіків. Приклад на наступному (рисунок 3.5) рисунку:

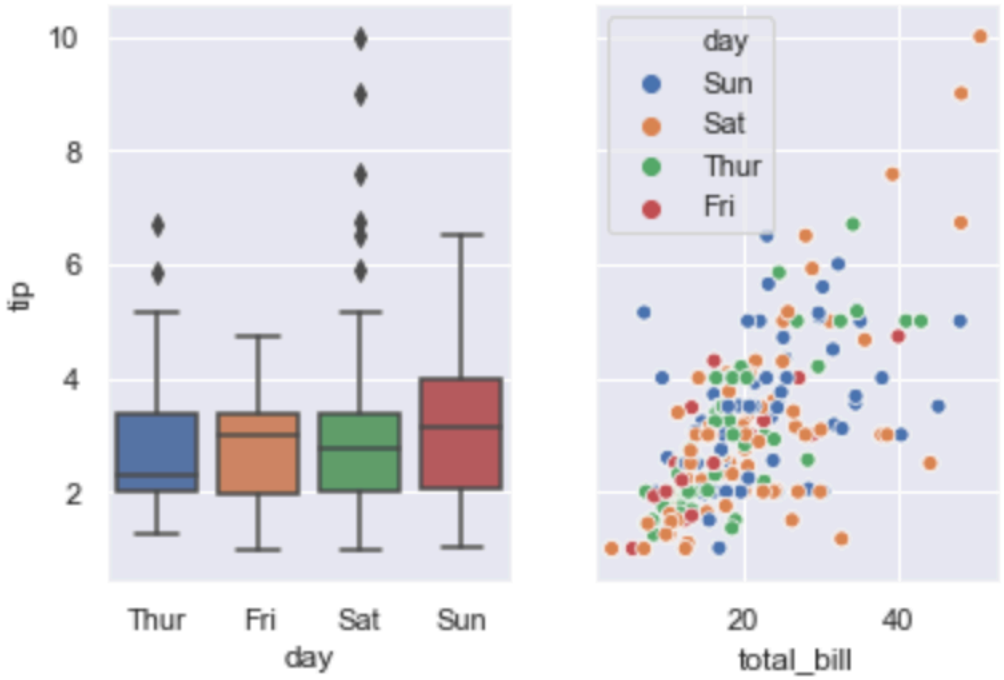


Рисунок 3.5 Приклад вигляду графіки у Seaborn

Matplotlib - це популярний пакет Python, який використовується для побудови графіків. Matplotlib почався як проект на початку 2000-х років, частково з використанням Python для візуалізації електронних сигналів в мозку пацієнтів з епілепсією. Творець Matplotlib, Джон Д. Хантер, був нейробіологом. Він шукав спосіб повторити можливості креслення MATLAB за допомогою Python. На додаток до створення Matplotlib, доктор Хантер був частиною групи засновників, яка створила Numfocus. Група Numfocus курирує деякі великі проекти Python, включаючи Matplotlib, NumPy, Pandas і Jupyter.

Далі (рисунок 3.6) можно побачити графіку Matplotlib.

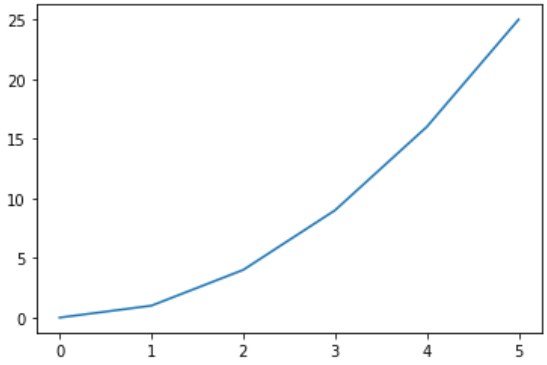


Рис. 3.6 Приклад графіки Matplotlib

Matplotlib корисний для створення статичних 2D-графіків, типу графіків, включених в наукові публікації та презентації. Майже будь-який сюжет, створений в Microsoft Excel, може бути створений за допомогою Matplotlib. Matplotlib також можна використовувати для створення 3D-сюжетів та анімації.

Остаточно, важлива складова дипломної роботи, котра не була досі описана – це середовище програмування.

У даній роботі використовується середовище Jupyter Notebook і не залучаються ніякі інші.

Документи блокнота (або «блокноти) - це документи, створені додатком Jupyter Notebook, які містять як комп'ютерний код (наприклад, python), так і елементи тексту фіксованої (абзаци, рівняння, малюнки, посилання тощо). Документи блокнота (рисунок 3.8) - це читаємі людиною документи, що містять опис аналізу та результати (малюнки, таблиці тощо), а також виконувані документи, які можна запустити для аналізу даних. Вигляд середовища (рисунок 3.7) на наступному рисунку.

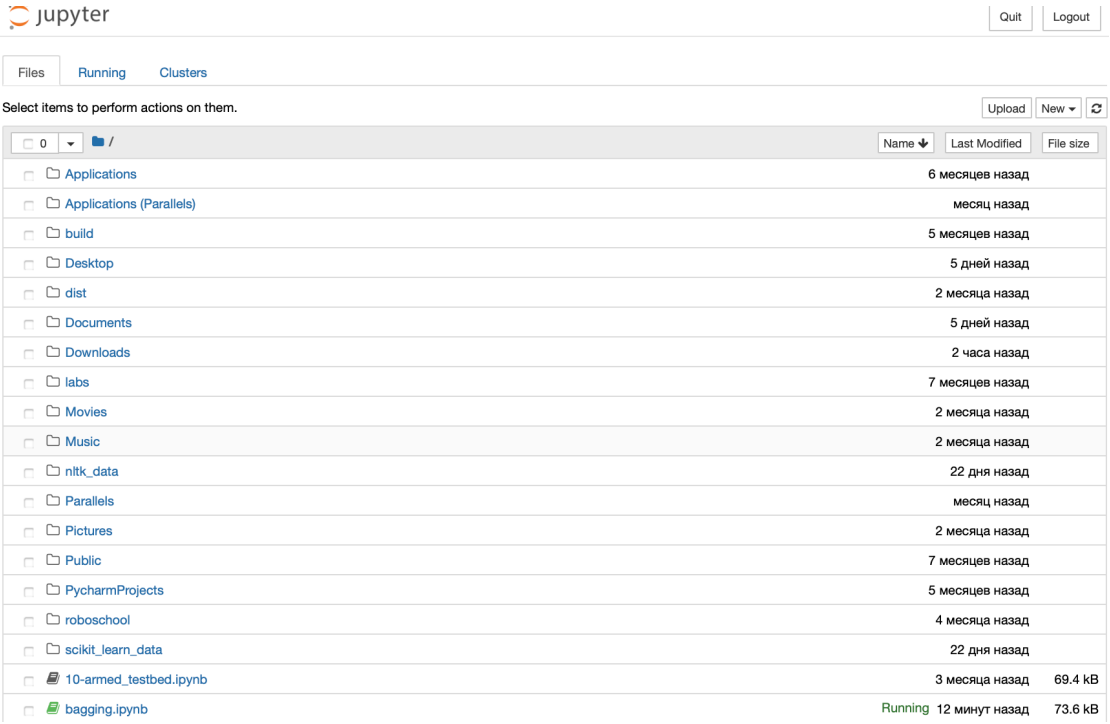


Рисунок 3.7 Вигляд додатку Jupyter Notebook

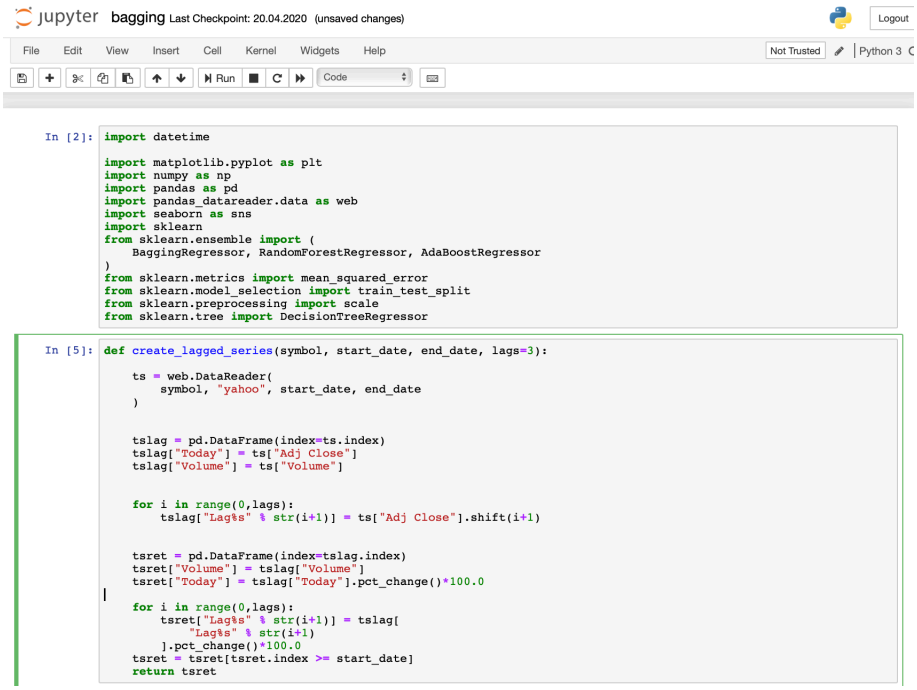


Рис. 3.8 Вигляд робочого «блокноту» з частиною коду дипломної роботи у Jupyter Notebook

Додаток Jupyter Notebook - це сервер-клієнтську програму, яка дозволяє редагувати і запускати документи ноутбука через веб-браузер. Додаток Jupyter Notebook може бути запущено на локальному комп'ютері і не вимагає доступу в Інтернет (як описано в цьому документі) або може бути встановлено на віддаленому сервері і доступно через Інтернет.

Крім відображення / редагування / запуску документів записної книжки, в додатку Jupyter Notebook є «Панель інструментів» (Notebook Dashboard), «панель управління», що показує локальні файли і дозволяє відкривати документи записної книжки або закривати їх ядра.

Ядро ноутбука - це «обчислювальний двигун», який виконує код, що міститься в документі Notebook. Ядро ipython, згадане в цьому керівництві, виконує код Python. Існують ядра для багатьох інших мов (офіційні ядра).

Коли ви відкриваєте документ Notebook, відповідне ядро запускається автоматично. Коли записна книжка виконується (по осередках або за допомогою меню Cell -> Run All), ядро виконує обчислення і видає результати. Залежно від типу обчислень ядро може споживати значні ресурси ЦП і ОЗУ. Зверніть увагу, що ОЗУ не звільняється, поки ядро не вимкнеться.

Важливо розуміти, що запускати програму, представлену у дипломній роботі необов’яхково локально на персональному комп’ютері.

Аналогічно Jupyter Notebook існує багато різних варіантів, з єдиною різницею у тому, що ви використовуєте не власні обчислювальні можливості, а «чужі», наприклад безкоштовні сервера, предоставлені Google. Такою заміною Jupyter Notebook є, наприклад, Google Collaboratory Notebook. Усі можливості локлального середовища тут представлені, обчислювальні можливості предоставляються компанією. Єдине обмеження та відмінність від локального середовища у тому, що локально ви не

обмежені у часі виконная програми, а у Google Collab (рисунок 3.9) надається 24 години.

Тобто, программа, котра виконуватиме обчислення більше ніж 24 години поспіль – не завершить свою роботу. Але можливо розбити обчислення на декілька частин та запускати послідовно.

Рис 3.9 Код програми з попереднього рисунка але у середовищі Google Collab

Таким чином, питання обчислювальних можливостей у даній дипломній роботі неважливе. Адже, будь яка машина здатна виконати програму, користуючись Google Collab, де замість її обчислювальних можливостей використовуватимуться чужі сервера. Цей варіант потребує підключення до інтернету. Якщо підключення до інтернету немає, тоді программа запускається локально на машині у середовищі Jupyter Notebook.

Отже, неважливо чи є підключення до інтернету чи ні, дозволяють технічні характеристики комп'ютера або ні – неважливо, програму виконати все одно вдасться та отримати ті ж самі результати прогнозу.

3.3 Естіматори

У функції __main__ (рисунок 3.10) встановлюються параметри. Спочатку визначається випадкове початкове число, щоб зробити код репліцируємим в інших робочих середовищах. n_jobs контролює кількість ядер процесора, використовуваних у бегінгу і рандом форест. Бустинг распарраралелити не можна, тому цей параметр не використовується для AdaBoost.

n_estimators визначає загальну кількість оцінок для використання в графіку MSE, в той час як step_factor контролює, наскільки роздріблеені розрахунки, шляхом покрокового виконання кількості оцінок. В цьому випадку axis_step дорівнює 1000/10 = 100. Тобто буде виконано 100 окремих обчислень для кожного з трьох методів ансамблю в фінальному розрахунку.

```
In [9]: if __name__ == "__main__":
            random_state = 42
            n_jobs = 4
            n_estimators = 1000
            step_factor = 10
            axis_step = int(n_estimators/step_factor)
```

Рисунок 3.10 Список параметрів зі значеннями для естіматора

Тут буде використано 4 ядра, що підвищить швидкість виконання. Для справедливого порівняння, цей параметр залишиться таким і для всіх інших моделей.

Далі, потрібно працювати з датасетом (рисунок 3.11). Наступний код завантажує ціни AMZN за десять років і перетворює їх в часовий ряд з лагами Пропущені значення відкидаються (наслідок процедури запізнювання), а дані масштабуються так, щоб вони існували між -1 і +1 для зручності порівняння. Ця остання процедура є звичайною в машинному навчанні та допомагає зіставити характеристики з великими відмінностями в абсолютних розмірах:

```
start = datetime.datetime(2010, 1, 1)
end = datetime.datetime(2020, 3, 31)
amzn = create_lagged_series("AMZN", start, end, lags=3)
amzn.dropna(inplace=True)
```

Рисунок 3.11 Список параметрів зі значеннями для датасету

Отже, будуть використані дані за період з 1 січня 2010 року до 31 березня 2020 року, що в свою чергу дає близько 10000 об'єктів у часовому ряді.

Головним елементом є ціна сьогодні. Вона пов'язана з ціною за вчора (Lag1), позавчора (Lag2) та за 3 дні до цього (Lag3) (рисунок 3.12). Характеристики самих даних можно побачити у розділі про вибірку.

```
X = amzn[["Lag1", "Lag2", "Lag3"]]
y = amzn["Today"]
X = scale(X)
y = scale(y)
```

Рисунок 3.12 Перетворення завантаженої вибірки

У машинному навчанні створюється модель, яка є не чим іншим, як алгоритмом, в якому деякі параметри повинні бути змінені таким чином, щоб він міг добре працювати в додатку, тобто він міг передбачити значення, які хтось хоче.

Як можна змінити ці параметри так, щоб вони могли добре працювати?

Можемо навчати модель, використовуючи дані, які називатимемо даними навчання або навчальними наборами. Навчальні дані - це ті дані, які вже мають фактичне значення, яке мала прогнозувати модель, і, таким чином, алгоритм змінює значення параметрів для обліку в навчальному наборі.

Але як дізнатися, що після тренування модель в цілому хороша?

Для цього використовуються тестові дані / набір тестів, які в основному представляють собою різні дані, для яких відомі значення, але ці дані ніколи не показувалися моделі раніше. Таким чином, якщо модель після

навчання також показує хороші результати на тестовому наборі, то можна сказати, що модель машинного навчання хороша.

Якщо модель не перевірена і зроблена так, що вона добре працює з даними навчання, тоді параметри будуть такими, що вони будуть досить хорошими, щоб передбачити цінність даних, які були в наборі навчання. Це не узагальнення. Це називається overfitting – перенавчання.

Таким чином, ми не збираємося робити марну модель, яка може бути доброю лише для тренувального набору і недостатньо універсальна.
Отже, набір тестів і навчальний набір важливі для поліпшення моделі машинного навчання.

Дані розбиті на навчальний набір і тестовий, при цьому 70% даних складають тренувальні дані, а інші 30% виконують тестовий набір (рисунок 3.13). Але, матимемо на увазі, що дані фінансових часових рядів послідовно корелюються, і тому ця процедура внесе деяку помилку, не врахувавши її. Однак це не є критичним для порівняння моделей у цій дипломній роботі.

```
In [22]:  X_train, X_test, y_train, y_test = train_test_split(
              X, y, test_size=0.3, random_state=random_state)
```

Рисунок 3.13 Розбиття часового ряду на тренувальний та тестовий

У наступних масивах NumPy (описано у розділі про бібліотеки) зберігається кількість оцінок на кожному кроці, а також фактично пов'язана MSE для кожного з трьох методів ансамблю. Всі вони (рисунок 3.14) спочатку встановлені на нуль і заповнюються згодом.

```
In [23]:  estimators = np.zeros(axis_step)
          bagging_mse = np.zeros(axis_step)
          rf_mse = np.zeros(axis_step)
          boosting_mse = np.zeros(axis_step)
```

Рисунок 3.14 Початкові значення помилки для усіх моделей

На цьому етапі усі підготовчі процедури виконані. Отримано фінансовий часовий ряд з корельованими даними, який виглядає наступним чином:

А) Наглядно

```
In [21]: amzn
```

Out[21]:

Date	Volume	Today	Lag1	Lag2	Lag3
2010-01-08	9830500	2.707696	-1.701323	-1.811569	0.589999
2010-01-11	8779400	-2.404139	2.707696	-1.701323	-1.811569
2010-01-12	9096300	-2.271506	-2.404139	2.707696	-1.701323
2010-01-13	10723200	1.382020	-2.271506	-2.404139	2.707696
2010-01-14	9774900	-1.363180	1.382020	-2.271506	-2.404139
...	...	...	...	...	...
2020-03-25	6479100	-2.796764	1.958663	3.073522	-1.852280
2020-03-26	6221300	3.693316	-2.796764	1.958663	3.073522
2020-03-27	5387900	-2.832539	3.693316	-2.796764	1.958663
2020-03-30	6126100	3.360348	-2.832539	3.693316	-2.796764
2020-03-31	5123600	-0.724559	3.360348	-2.832539	3.693316

Рисунок 3.15 Читаємий вигляд датасету

Б) Як передається у моделі (значення для Тренувального та Тестувального набору)

```
[[-0.92284778 -0.97788636  0.2358977 ]
 [ 1.30583414 -0.92214039 -0.97897292]
 [-1.27810893  1.30728683 -0.92320335]
 ...
 [ 1.80404789 -1.47605149  0.92825757]
 [-1.4946575   1.80566719 -1.47734892]
 [ 1.63573863 -1.49414133  1.80575888]]
 [ 1.30581168 -1.27846655 -1.21141398 ... -1.49504321  1.63575895
-0.42935814]
```

Рисунок 3.16 Вигляд передаваємих у моделі значень

Перший метод ансамблю (рисунок 3.17), який повинен використовуватися, є процедурою bagging. Код перебирає загальна кількість оцінювачів (в даному випадку від 1 до 1000, з розміром кроку 10), визначає модель ансамблю з правильною базовою моделлю (в даному випадку це дерево рішень регресії), підходить для навчання даних, а потім розраховує середньоквадратичну помилку (MSE) на тестових даних . Ця MSE тоді додається до масиву MSE для bootstrap aggregating.

```
In [*]:     for i in range(0, axis_step):
                print("Bagging Estimator: %d of %d..." % (
                    step_factor*(i+1), n_estimators)
                )
                bagging = BaggingRegressor(
                    DecisionTreeRegressor(),
                    n_estimators=step_factor*(i+1),
                    n_jobs=n_jobs,
                    random_state=random_state
                )
                bagging.fit(X_train, y_train)
                mse = mean_squared_error(y_test, bagging.predict(X_test))
                estimators[i] = step_factor*(i+1)
                bagging_mse[i] = mse
```

```
Bagging Estimator: 10 of 1000...
Bagging Estimator: 20 of 1000...
Bagging Estimator: 30 of 1000...
Bagging Estimator: 40 of 1000...
Bagging Estimator: 50 of 1000...
Bagging Estimator: 60 of 1000...
Bagging Estimator: 70 of 1000...
Bagging Estimator: 80 of 1000...
Bagging Estimator: 90 of 1000...
Bagging Estimator: 100 of 1000...
Bagging Estimator: 110 of 1000...
Bagging Estimator: 120 of 1000...
Bagging Estimator: 130 of 1000...
Bagging Estimator: 140 of 1000...
Bagging Estimator: 150 of 1000...
Bagging Estimator: 160 of 1000...
Bagging Estimator: 170 of 1000...
Bagging Estimator: 180 of 1000...
Bagging Estimator: 190 of 1000...
Bagging Estimator: 200 of 1000...
Bagging Estimator: 210 of 1000...
```

Рисунок 3.17 Робота естіматора bagging в процесі

Той же підхід застосовується для Random Forest (рисунок 3.18). Оскільки випадковий ліс неявно використовує дерево регресії в якості своїх базових оцінювачів (естіматорів), немає необхідності вказувати його в конструкторі ансамблю. Бібліотека sklearn разом з NumPy та Pandas дозволяє мінімізувати кількість коду, необхідного для опсиання естіматора.

```
In [11]:    for i in range(0, axis_step):
                print("Random Forest Estimator: %d of %d..." % (
                    step_factor*(i+1), n_estimators)
                )
                rf = RandomForestRegressor(
                    n_estimators=step_factor*(i+1),
                    n_jobs=n_jobs,
                    random_state=random_state
                )
                rf.fit(X_train, y_train)
                mse = mean_squared_error(y_test, rf.predict(X_test))
                estimators[i] = step_factor*(i+1)
                rf_mse[i] = mse
```

```
Random Forest Estimator: 10 of 1000...
Random Forest Estimator: 20 of 1000...
Random Forest Estimator: 30 of 1000...
Random Forest Estimator: 40 of 1000...
Random Forest Estimator: 50 of 1000...
Random Forest Estimator: 60 of 1000...
Random Forest Estimator: 70 of 1000...
Random Forest Estimator: 80 of 1000...
Random Forest Estimator: 90 of 1000...
Random Forest Estimator: 100 of 1000...
Random Forest Estimator: 110 of 1000...
Random Forest Estimator: 120 of 1000...
```

Рисунок 3.18 Робота естіматора Random Forest в процесі

Як видно, обидва естіматора (Bagging та Random Forest) описані однаково, з різницею у регресорі. Для Bagging регресор – це DecisionTreeRegressor, а для Random Forest – RandomForestRegressor. Ці два регресори і є причина, чому написання програми на мові програмування Python з пакетом sklearn є оптимальним для задачі дипломної роботи та оптимальним для задач машинного навчання взагалі. У цьому короткому записі повністю описана робота алгоритму, формально описану у розділі 1 для кожної з моделей окремо.

Таким чином, на цьому етапі залишається остання модель – AdaBoost. Вона відрізняється від попередніх, оскільки належить до класу бустингу, не може бути розпаралелена і використовує регрессор як і бегінг, але зовсім по іншому. Як само, описано у розділі 1.

Аналогічно для алгоритму бустингу AdaBoost (рисунок 3.19), хоча n_jobs не присутній через відсутність паралелізму методів бустингу. Швидкість навчання або коефіцієнт усадки (λ) був встановлений на 0,01.

Коригування цього значення дуже впливає на абсолютну MSE, розраховану для кожного підсумкового значення оцінки.

```
In [*]: for i in range(0, axis_step):
            print("Boosting Estimator: %d of %d..." % (
                step_factor*(i+1), n_estimators)
            )
            boosting = AdaBoostRegressor(
                DecisionTreeRegressor(),
                n_estimators=step_factor*(i+1),
                random_state=random_state,
                learning_rate=0.01
            )
            boosting.fit(X_train, y_train)
            mse = mean_squared_error(y_test, boosting.predict(X_test))
            estimators[i] = step_factor*(i+1)
            boosting_mse[i] = mse
```

```
Boosting Estimator: 10 of 1000...
Boosting Estimator: 20 of 1000...
Boosting Estimator: 30 of 1000...
Boosting Estimator: 40 of 1000...
Boosting Estimator: 50 of 1000...
Boosting Estimator: 60 of 1000...
Boosting Estimator: 70 of 1000...
```

Рисунок 3.19 Робота естіматора AdaBoost в процесі

У коді, відповідаючому описанню естіматора AdaBoost можна побачити кардинальні відмінності від попередніх двух. Відсутність n_jobs (тобто кількості ядер) та поява коефіцієнту learning_rate (Лямбда) обумовлені кардинальною різницею ідей алгоритмів, що, в свою чергу, обумовлено належністю до різних класів (Ансамблі для попередніх, Бустинг для цього) описаних у розділі 1.

Коли робота кожної моделі завершена, залишається оцінити результати (рисунок 3.20), представивши їх у наглядному вигляді за допомогою бібліотек Matplotlib та Seaborn.Останній фрагмент коду просто будує ці масиви один проти одного, використовуючи Matplotlib, але з колірною схемою Seaborn за замовчуванням, яка більш приваблива, ніж значення за замовчуванням Matplotlib.

```
In [40]:  plt.figure(figsize=(8, 8))
          plt.title('Порівняння бегінгу, Random Forest та бустингу')
          plt.plot(estimators, bagging_mse, 'b-', color="red", label=' Bootstrap aggregating')
          plt.plot(estimators, rf_mse, 'b-', color="green", label='Random Forest')
          plt.plot(estimators, boosting_mse, 'b-', color="black", label='AdaBoost')
          plt.legend(loc='upper right')
          plt.xlabel('Кількість естіматорів')
          plt.ylabel('Значення середньквадратичної помилки')
          plt.show()
```

Рисунок 3.20 представлення результатів роботи моделей а наглядному вигляді.

4 ПОРІВНЯННЯ

4.1 Опис критеріїв для порівняння

Головним критерієм у порівнянні буде значення середньоквадратичної помилки (mean squared error (MSE)).

Мета задачі машинного навчання - мінімізувати помилку, яка визначається функцією втрат. І у кожного типу алгоритму є різні способи вимірювання помилки.

Припустимо, ми отримали 2 функції втрат. Обидві функції будуть мати різні мінімуми. Тому, якщо оптимізувати неправильну функцію втрат, отримаємо неправильне вирішення, яке є оптимальною точкою або оптимізованим значенням ваги у функції втрат. Або можна сказати, що вирішується неправильна проблема оптимізації. Тому потрібно знайти підходящу функцію втрат, яку будемо мінімізувати.

Помилка (рисунок 4.1) повинна зменшуватися в міру того, як ми збільшуємо наші вибіркові дані, оскільки розподіл наших даних стає все більш і більш вузьким (посилаючись на нормальний розподіл). Чим більше у нас даних, тим менше помилка. Цій ідеї відповідає MSE.

$$\mathbf{L} = \frac{1}{\mathrm{N}}\left[\,\Sigma(\hat{\mathrm{Y}} - \mathrm{Y})^2\right]$$

Рисунок 4.1 Функція MSE

4.2 Висновки після порівняння

Вихідні дані наведені на наступних рисунках. Чітко видно, як збільшення числа оцінок для методів, заснованих на бутстрепі (Bagging та Random Forest), в кінцевому підсумку призводить до того, що MSE «заспокоюється» і стає майже ідентичним між ними. Проте, для алгоритму бустингу AdaBoost видно, що в міру того, як число оцінювачів збільшується більш ніж на 100 алгоритм не покращує значення помилки, але й не перенавчається. Хоча результат роботи адаптивного бустингу і є гіршим, в абсолютному значенні це різниця у 0.125, що не є критичним. Але швидкість роботи та краща точність у бегінгу не залишає шансів AdaBoost.

Це період з 2015 по 2020 рік. Період стабільного росту акцій до моменту пандемії коронавіруса. Якщо відійти від моделей та подивитися з іншого боку, цей період стабільності для компанії, що означає невелику кількість значних стрибків акцій. На такому датасеті вдасться досягти кращого асболютного значення помилки, але реальної практичної користі вона не нестиме, адже моделі не будуть знати про можливі стрибки (аутлаєри), котрі з'являлися у період з 2010 по 2015 рік.

Наступні графіки (рисунок 4.2 – 4.5) дозволяють зрозуміти, що обрані параметри для остаточних моделей на даних за 10 років (найвпливоішим є кількість естіматорів) є оптимальним.

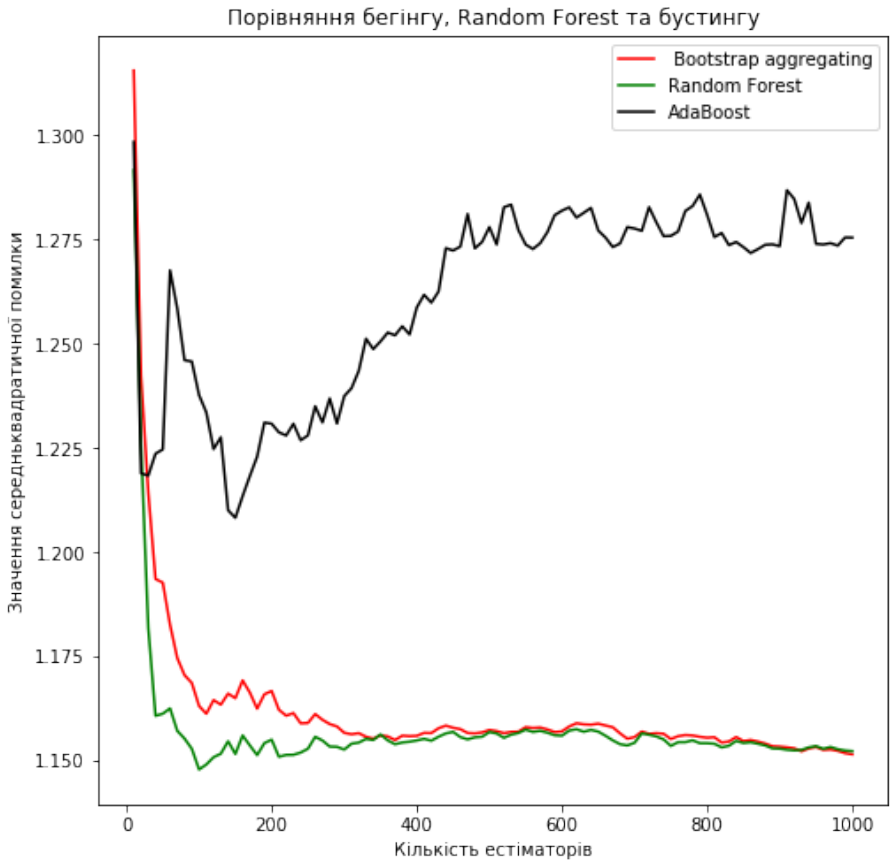


Рисунок 4.2 Значення помилки для 1000 естіматорів на даних за 5 років

Тобто, на середній кількості даних, при оптимальних параметрах моделей, Bagging показує найкращі результати, Random Forest займає другу позицію з незначним відставанням у точності прогнозу, а адаптивний бустинг займає третю позицію, оскільки значення помилки для нього помітно більше.

На наступному рисунку показано результати прогнозу для інших параметрів моделей, на даних за 5 років.

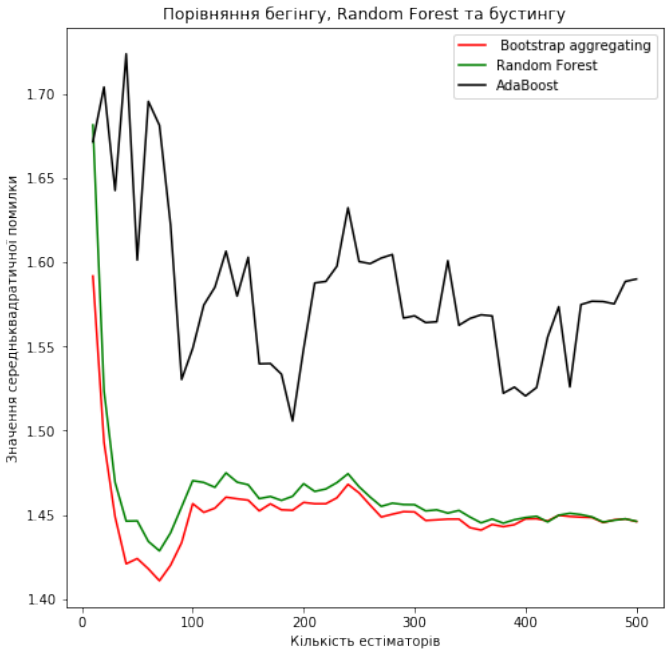


Рисунок 4.3 Значення помилки для 500 естіматорів на даних за 5 років

Значення помилки при таких параметрах є помітно гіршими для кожного алгоритму, а для кількості естіматорів 1500 значення помилки майже не відрізняється від помилки для 1000, проте час на навчання кардинально більший. Ці міркування приводять до того, що збільшення кількості естматорів після 1000 не ведуть до зменшення помилки, але значно сповільнюють процесс навчання.

Наступний рисунок показує значення помилки для моделей на даних за період 2010 – 2020 роки (період стабільного росту акцій). Значення помилки на ньому стабілізується після 200 етсіматорів та залишається таким до 1000.

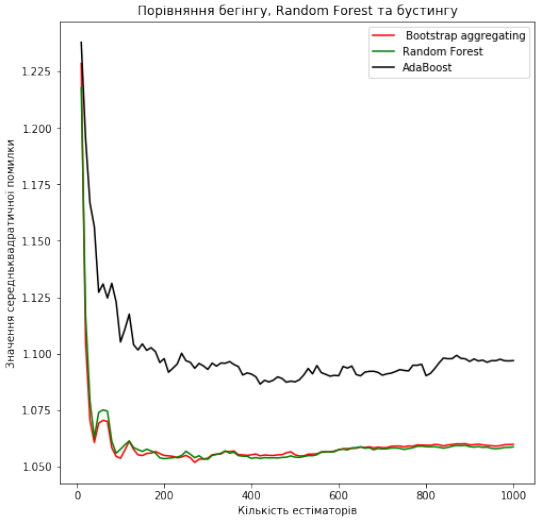


Рисунок 4.4 Значення помилки для 1000 естіматорів на даних за 10 років

Але, що справді важливо – це включити у модель дані за період кризи 2007-2008 років з даними іпотечної кризи 2006 року. Саме ці дані дозволять створити актуальну на сьогодні модель, котра зможе давати ефективний прогноз акцій компаній у період можливої кризи після пандемії коронавірусу.

Остаточно, результати для набору даних за 20 років при оптимальних значеннях параметрів для кожної моделі.

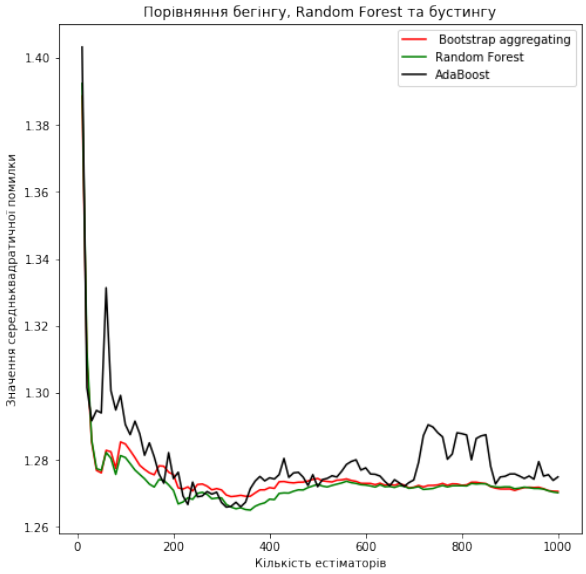


Рисунок 4.5 Значення помилки для 1000 естіматорів на даних за 20 років

Чітко видно як значення помилки стабілізуються згодом для бегінгу та Random Forest за незначним відставанням у абсолютному значені для бустингу AdaBoost.

При цьому, абсолютне значення помилки більше ніж у періоди стабільного росту (2010-2020 роки). Через те, що у модель тренування були включені дані за період кризи 2007-2008 років, абсолютне значення помилки для моделей більше за попередні результати на даних стабільного росту, але лише у цьому варінті моделі справді стають актуальними на сьогодні, оскільки вони готові до прогнозу у період кризи, що є необхідним у сьогоденних реаліях.

5 ФУНКЦІОНАЛЬНО-ВАРТІСТНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

5.1. Постановка задачі

Проводиться оцінка основних характеристик результатів та їх собівартість. Для отримання результатів використовувалась мова Python. Середовище розробки - Jupyter Notebook. Робота програми не залежить від технологій реалізації апаратного забезпечення та операційної системи. Нижче приведено аналіз різних підходів до використання моделей для прогнозу часового ряду.

5.2. Обґрунтування функцій та параметрів дослідження

Основні функції:

F_1 - кількість даних у вибірці; F_2 - вибір підходу до розв'язку задачі прогнозування; F_3 - вибір мови програмування

Функція F_1: а) 1000об’єктів у вибірці; б) 10000 об’єктів у вибірці; в) 100000 об’єктів у вибірці.

Функція F_2: а) Підхід з використанням bootstrapaggregation; б) стандартний підхід.

Функція F_3: а) Вибір мови програмування Python; б) Вибір мови програмування C++; в) Вибір мови програмування Mathlab;

Знизу (рисунок 5.1) зображено відповідну морфологічну карту системи:

1000	10000	100000

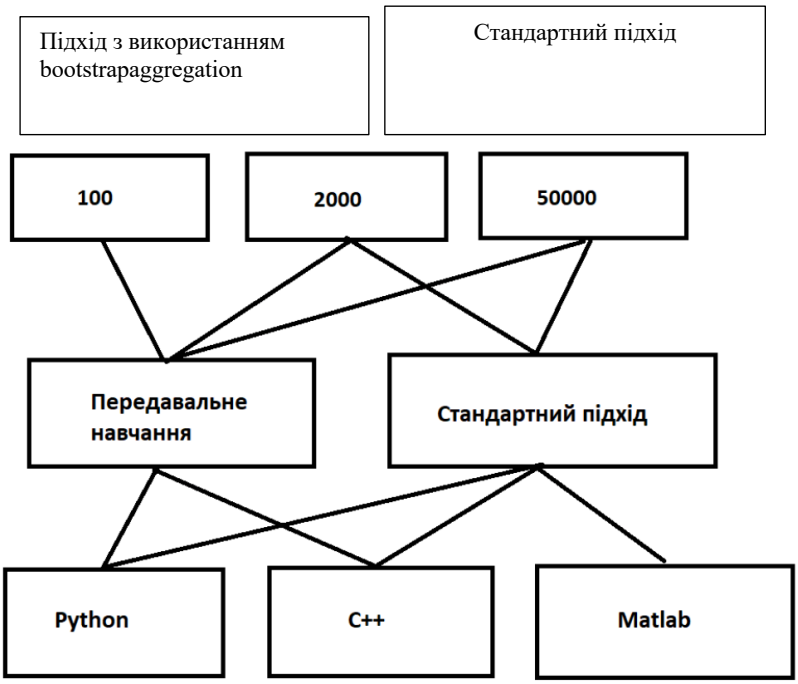


Рисунок 5.1 - Морфологічна карта

Позитивно-негативна матриця варіантів основних функцій:

Основні функції	Варіанти реалізацій	Переваги	Недоліки
F_1	А	Швидко, не потрібне програмне забезпечення	Для моделі прогнозу може бути замало даних для навчання
F_1	Б	Достатня кількість даних для моделей. Можна зберігати як локально так і на сервері	Необхідне програмне забезпечення

F_1	В	Можна швидко отримати усі дані, якщо зберігати на сервері.	Необхідне програмне забезпечення.
F_2	А	Не потрібні потужні характеристики компьютеру. Навчання проходить надзвичайно швидко	Обмеженість у виборі моделей для прогнозу
F_2	Б	Гнучкість відносно архітектури	Потрібні потужні характеристики компьютеру.
F_3	А	Вже готові бібліотеки та алгоритми. Ідеально підходить для машинного навчання.	На великих обсягах вибірки навчання займатиме набагато більше часу
F_3	Б	Висока швидкість	Створення спеціальних алгоритмів навчання є надзвичайно складне та довге завдання. Власний алгоритм може виявитись повільніший ніж на Python

F_3	B	Зручність для користувача	Висока ціна забезпечення.

Таблиця 5.1 Позитивно-негативна матриця

Тепер, за наявності позитивно-негативної матриці можна робити висновки щодо доцільності використання одних варіантів та недоцільності використання інших:

Для характеризації досліджень пропонуємо такі параметри:

X1 — точність моделі на небачених даних (%спрогнозованих значень з точністю менше прийнятної);

X2 - навчання моделі прогнозу (час, за який відбулося навчання);

X3 - складність освоєння теоретичної бази(час на освоєння теоретичної бази);

X4 — Час на освоєння мови програмування (Час на вивчення мови програмування);

X5 - складність освоєння бібліотеки sklearn (Час на освоєння бібліотеки sklearn);

X6 — Оперативна память (кількість оперативної пам’яті для роботи);
Гірші, середні та кращі показники параметрів вибираються на основі вимог замовника та умов перебігу дослідження, їх наведено у таблиці 2:

Умовні позначення	Одиниці виміру	Мінімальне значення	Середнє значення	Максимальне
X1	%	0	60	100
X2	Хв.	0.5	1	10

X3	Год.	10	15	30
X4	Год.	50	70	120
X5	Год.	4	6	15
X6	ГБ	2	4	32

Таблиця 5.2 Система параметрів додатку

Графічні харатеристики описаних вище параметрів:

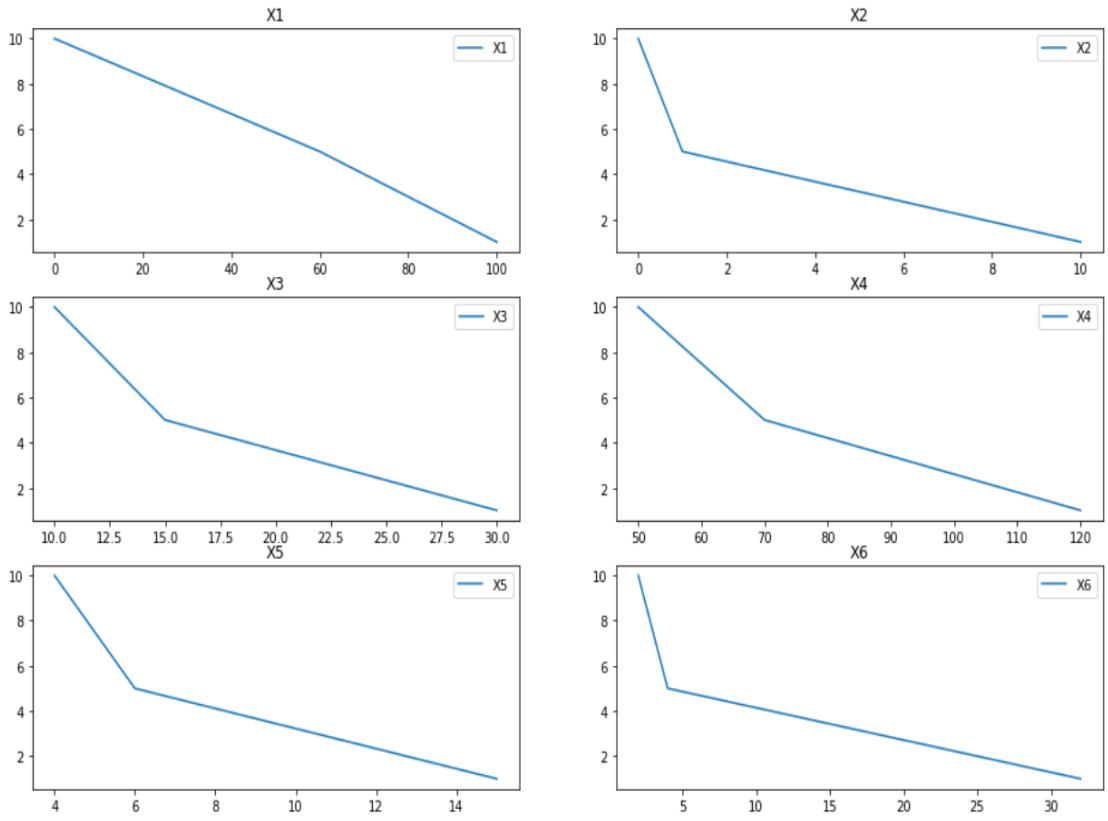


Рисунок 5.2 Значення параметрів

Вагомість параметрів визначається методом попарного їх порівняння на основі результатів ранжування експертами та попарного порівняння параметрів. Наведемо результати експертного ранжування(таблиця 3):

Умовне позначення	Одиниці вимірювання	Ранг експерта 1	Ранг експерта 2	Ранг експерта 3	Ранг експерта 4	Ранг експерта 5	Ранг експерта 6	Ранг експерта 7	Сума рангів R_i	Відхилення Δ_i	$\Delta^2{}_i$
X1	Год.	1	2	1	1	1	3	1	10	-14.5	210.25
X2	Год.	3	1	2	3	3	1	2	21	-9.5	90.25
X3	Год.	2	3	3	2	2	2	3	20	-7.5	56.25
X4	Год.	6	6	5	6	6	6	4	39	14.5	210.25
X5	Год.	5	4	6	4	5	4	5	26	8.5	72.25
X6	ГБ	4	5	4	5	4	5	6	31	8.5	72.25
Разом		21	21	21	21	21	21	21	147	0	711.5

Табл. 5.3 Результат оцінки параметрів

Найменший ранг – найкращий.

Параметри	1	2	3	4	5	6	7	Кінцева оцінка	Числове значення
X1 та X2	<	>	<	<	<	>	<	<	1.5

X1 та X3	<	<	<	<	<	>	<	<	1.5
X1 та X4	<	<	<	<	<	<	<	<	1.5
X1 та X5	<	<	<	<	<	<	<	<	1.5
X1 та X6	<	<	<	<	<	<	<	<	1.5
X2 та X3	>	<	<	>	>	<	<	<	1.5
X2 та X4	<	<	<	<	<	<	<	<	1.5
X2 та X5	<	<	<	<	<	<	<	<	1.5
X2 та X6	<	<	<	<	<	<	<	<	1.5
X3 та X4	<	<	<	<	<	<	<	<	1.5
X3 та X5	<	<	<	<	<	<	<	<	1.5
X3 та X6	<	<	<	<	<	<	<	<	1.5
X4 та X5	>	>	<	>	>	>	<	>	0.5
X4 та X6	>	>	>	>	>	>	<	>	0.5
X5 та X6	>	<	>	<	>	<	<	<	1.5

Продовження Табл. 5.4 Попарне зрівняння параметрів

За даними таблиці 3 визначимо коефіцієнт конкордації за формулою:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12*711.5}{49*(216-6)} \approx 0.83 > 0.67 \qquad (5.1)$$

— достовірне, виходячи із даної нерівності.

Аналіз рівня якості варіантів реалізації функцій

Розрахунок вагомості параметрівзанотуємо у таблицю 5, показники рівня якості у таблицю 6:

	X1	X2	X3	X4	X5	X6	b_i^1	K_i^1	b_i^2	K_i^2	b_i^3	K_i^3
X1	1	1.5	1.5	1.5	1.5	1.5	8.5	0.236	49.75	0.25	272.87	0.25
X2	0.5	1	1.5	1.5	1.5	1.5	7.5	0.208	41.75	0.21	227.1	0.21
X3	0.5	0.5	1	1.5	1.5	1.5	6.5	0.180	34.75	0.175	188.87	0.17
X4	0.5	0.5	0.5	1	0.5	0.5	3.5	0.097	19.75	0.099	109.12	0.1
X5	0.5	0.5	0.5	1.5	1	1.5	5.5	0.152	28.75	0.141	157.12	0.14
X6	0.5	0.5	0.5	1.5	0.5	1	4.5	0.12	23.75	0.119	130.87	0.12

Табл. 5.5 Розрахунок вагомості параметрів

На основі порівняльного аналізу варіантів реалізації основних функцій по їх перевагам та недолікам можна виключити варіанти F_1 А і В , F_3 Б і В, тоді варіанти, які залишилися:

F_1 Б -> F_2 А -> F_3 А

F_1 Б -> F_2 Б -> F_3 А

Основні функції	Варіанти реалізацій	Параметри	Абсолютне Значення	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	Б	X1	18	8.33	0.04	0.33
F_2	А	X1	60	5	0.1	0.5
		X2	5.5	2.5	0.06	0.15
F_2	Б	X1	81	2.54	0.1	0.25
		X2	10	1	0.1	0.1
F_3	А	X3	22.5	2.5	0.05	0.13
		X4	95	2.5	0.15	0.38
		X5	10.5	2.5	0.07	0.18

Табл. 5.6 Розрахунок показників якості

$$K1 = 0.33+0.5+ 0.15 +0.13+0.38+0.18 = 1.67 \quad (5.2)$$

$$K2 = 0.33+ 0.25+ 0.1 + 0.13+0.38+0.18 = 1.37 \quad (5.3)$$

Отже, перший варіант, що передбачає 10000 об’єктів у вибірці з bootstrap на Python дає більший результат. Тому віддаємо йому перевагу.

5.3 Економічний аналіз варіантів розробки ПП

Обидва варіанти включають в себе три окремих етапи:

1 підготовки даних

2 навчання моделей:

2.1 З bootstrap aggregating

2.2 Без бутстрепінгу

3 розробка програмного продукту

Для завдання 1 (Алгоритм складності 3, ступінь новизни Г, вид використаної інформації БД)

$$T_p = 12,\ К_п = 0.3,\ К_ст = 0.7,\ Кст.м = 1.2 \quad (5.4)$$

$$T_1 = 12*0.3*0.7*1.2 = 3.024 \quad (5.5)$$

Для завдання 2 (при реалізації варіанту 1), (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ)

$$T_p = 64,\ К_п = 2.02,\ К_ст = 0.8,\ Кст.м = 1.6 \quad (5.6)$$

$$T_2^1 = 64*\ 2.02\ *\ \ 0.8\ *\ 1.6 = 165.478 \text{ людино-днів} \quad (5.7)$$

Для завдання 2 (при реалізації варіанту 2)), (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ)

$$T_p = 64,\ К_п = \ 2.02,\ К_ст = 0.8,\ Кст.м = 1.4 \quad (5.8)$$

$$T_2^1 = 64*\ 2.02\ *\ \ 0.8\ *\ 1.4 = 144.793 \text{ людино-днів} \quad (5.9)$$

Для завдання 3(Алгоритм складності 1, ступінь новизни В, вид використаної інформації БД)

$$T_p = 43,\ К_п = \ 1.35,\ К_ст = 0.6,\ Кст.м = 1.5 \quad (5.10)$$

$$T_3 = 43*1.35*0.6*1.5 = 52.24 \text{ людино-днів} \quad (5.11)$$

$$T1 = (3.024 + 165.478 + 52.24) * 8 = 1765 \text{ людино-годин} \quad (5.12)$$

$$T2 = (3.024 + 144.79 + 52.24) * 8 = 1600 \text{ людино-годин} \quad (5.13)$$

В розробці та проведенні дослідження приймають участь 2 програмісти з окладом 20 000 грн

Зарплата розробників становить погодинно:

$$С = (20000+20000) / (2* 21 * 8) = 119 \text{ грн} \quad (5.14)$$

Зарплата поваріантно

$$С\_1 = 1765*119 = 210035 \text{ грн} \quad (5.15)$$

$$С\_2 = 1600*119 = 190400 \text{ грн} \quad (5.16)$$

Відрахування на соціальний внесок становить 22%:

$$С\_1^v = 0.22 * 210035= 46208 \text{ грн} \quad (5.17)$$

$$С\_2^v = 0.22 * 190400 = 41888 \text{ грн} \quad (5.18)$$

Визначаємо витрати на оплату однієї машино-години. З урахуванням заробітної плати програміста в розмірі 20000 грн з коефіцієнтом зайнятості 0.2, маємо

$$С\_г = 12*М*К\_З = 12 *20000* 0,2 = 48000 \text{ грн} \quad (5.19)$$

З урахуванням додаткової заробітної плати:

$$СЗП =СГ* (1+ КЗ) = 48000 * (1 + 0.2)=57600 \text{ грн} \quad (5.20)$$

Відрахування на соціальний внесок:

$$С\_ВІД= С\_ЗП * 0.22 = 57600 * 0,22 = 12672 \text{ грн.} \quad (5.21)$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн

$$С\_А = К\_ТМ* К\_А*Ц\_ПР = 1.15 * 0.25 * 10000 = 2875 \text{ грн.} \quad (5.22)$$

Витрати на ремонт та профілактику можна підрахувати:

$$С\_Р = К\_ТМ*Ц\_ПР * К\_Р = 1.15 * 10000 * 0.05 = 575 \text{ грн.,} \quad (5.23)$$

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т\_ЕФ =(Д\_К – Д\_В – Д\_С – Д\_Р) * t\_З*К\_В = (365 – 104 – 11 – 16) * 8 * 0.9 = 1684.8 \quad (5.24)$$

Тепер рахуємо витрати на оплату електроенергії (з урахуванням ПДВ): С_ЕЛ = Т_ЕФ* N_С* К_З* Ц_ЕН =1684.8 * 0,3 * 0.2 * 1.75 = 2586.8 грн. (5.25)

Накладні витрати рахуються наступним чином:

С_Н = Ц_ПР*0.67 = 10000· 0,67 =6700 грн (5.26)

Річні експлуатаційні витрати:

С_ЕКС = 57600 + 12672 + 2875 + 575 + 2586.8 + 6700 = 83008.8 грн (5.27)

Собівартість однієї машино-години ЕОМ становитиме:

С_М-Г = С_ЕКС/ Т_ЕФ = 83008.8 /1684.8 = 49.3 грн/год (5.28)

Витрати на оплату машинного часу складають в залежності від варіанту:

С_М^1 = 49.3 * 1765 = 87014.5 грн (5.29)

С_М^1 = 49.3 * 1600 =78880 грн (5.30)

Накладні витрати складають 67% від заробітної плати:

С_Н^1 = 0.67 * 210035 = 140723 грн (5.31)

С_Н^2 = 0.67 * 190400 = 127568 грн (5.32)

Таким чином, вартість розробки ПП та проведення дослідів складає:

С_1 = 210035 + 46208 + 87014.5 + 140723 = 483980.5 грн (5.33)

С_2 = 190400 + 41888 + 78880+ 127568 = 438736 грн (5.34)

Проведемо розрахунок техніко-економічного рівня:

К_ТЕР1 = 1.67/ 483980.5 = 0.35 * 10^-5 (5.35)

К_ТЕР2 = 1.37/ 438736 = 0.31 * 10^-5 (5.36)

Отже найбільш ефективним є перший варіант з коефіцієнтом техніко економічного рівня 0.35 * 10^-5

Висновки до розділу 5

За результатами проведеного функціонально-вартісного аналізу, можна зробити висновок, що з альтернатив, що залишилися після відбору, було отримано два варіанти.В результаті проведення функціонально-вартісного аналізу стало зрозуміло, що доцільним є використання першого варіанту реалізації продукту. Першому варіанту відповідає навчання моделей на вибірці розміром 10000об’єктів з використанням bootstrap aggregation на мові програмування Python. Таким чином, отримано моделі, якіз прийнятною точністю прогнозують часові ряди та не потребують великих об'ємів даних та обчислювальних ресурсів.

ВИСНОВКИ

В дипломній роботі виконано дослідження часового ряду (період економіної кризи 2007-2008 років, періоди стабільного росту акцій) та моделей для його прогнозування, зокрема алгоритмів, заснованих на bootstrap aggregating. Також, виконано порівняльну характеристику абсолютних значень MSE для моделей рызної природи: бустингу AdaBoost, Random Forest та bagging.

Проведено огляд алгоритмів прогнозування для реального фінансового часового ряду компанії Yahoo! Finance, їх характеристик та методів, та розглянуто варіанти їх реалізації та особливості їх застосування для цього ряду, а також необхідні для цього пакети мови програмування Python.

Розглянуто принципи побудови моделей та естіматорів. Також, розглянуто імплементацію цих моделей у програмну реалізацію у середовищі Jupyter Notebook.

Виконано роботу над датасетом згідно з прийнятими для машинного навчання канонами та описано, обгрунтовано чому цей датасет є хорошим набіром даних для виконаних моделей, які слабкі сторони кожного алгоритму проявляються на цій вибрці. Це ставить кожну модель прогнозування у справедливе порівняння з іншими.

Також, обгрунтовано актуальність моделей на сьогодні з можливістю економічної кризи через пандемію коронавірусу.

За результатами роботи було встановлено, що найкращого абсолютного значення помилик вдалось досягти алгоритмам Random Forest та Bootstrap Aggregating. Вони дають найменшу помилку, швидко навчаються (у порівнянні з бустингом).

Усі моделі є актуальним для прогнозування фінансових часових рядів як в періоди стабльного росту чи спадання, так і в періоди економічної кризи.

Використання Bootstrap Aggregating є найраціональнішим сьогодні через найменше абсолютне значення помилки та готовності до ефективного прогнозу в період можливої майбутньої економічної кризи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Efron, B. (1981) "Bootstrap methods: Another look at the jackknife", *The Annals of Statistics* **8** (1): 5-36
   URL: (https://projecteuclid.org/euclid.aos/1176344552) (дата звернення: 15.04.2020)
2. Breiman, L. (1996) "Bagging predictors", *Machine Learning* **24** (2): 123-140
   URL: (https://link.springer.com/article/10.1007/BF00058655) (дата звернення: 20.04.2020)
3. Wikipedia (2016) Wikipedia: Random Forest, https://en.wikipedia.org/wiki/Random_forest
   URL: (https://en.wikipedia.org/wiki/Random_forest) (дата звернення: 20.05.2020)
4. Фінансовий звіт Yahoo! Finance
   URL:(https://finance.yahoo.com) (дата звернення: 18.04.2020)
5. https://uk.wikipedia.org/wiki/Python (дата звернення: 21.05.2020)
6. https://pythonguide.rozh2sch.org.ua/ (дата звернення: 18.05.2020)
7. https://habr.com/ru/company/wunderfund/blog/316826/ (дата звернення: 18.05.2020)

ДОДАТОК А ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДОПОВІДІ

Покращення прогнозування часових рядів за допомогою підходу з використанням bootstrap aggregation

Виконав: Островерхий Антон, КА-61

Науковий керівник: к.ф.-м.н. доц. Яковлева А.П.

Київ 2020

Актуальність

- Актуальність дослідження: прогнозування фінансового часового ряду компанії Yahoo Finance за великий проміжок часу (більше 10 років) з можливістю включення найактуальніших на сьогодні даних у ряд. Представлені моделі показують хороші результати, мінімізуючи помилку, при цьому час на навчання мінімальний, навіть на реальних даних за 10 років.

Вступ

- Предмет дослідження: моделі Bagging, Random Forest та AdaBoost для прогнозування часового ряду, засновані на бустрепінгу та бустингу, їх особливості та застосування на реальному фінансовому часовому ряді.
- Методи дослідження: застосовані моделі прогнозування різної природи для часового фінансового ряду, виконані на мові програмування Python у середовищі Jupyter Notebook з використанням пакету sklearn.
- Об’єкт дослідження: набір фінансових даних компанії Yahoo! Finance з 1998 року.
- Мета: побудувати моделі прогнозу часового фінансового ряду та порівняти значення помилки для різних підходів

Постановка задачі

- В дипломній роботі поставлені для виконання такі задачі:

- Побудувати три моделі різної природи (AdaBoost, Bootstrap Aggregating, Random forest)
- Порівняти результати роботи моделей.
- Дослідити Bootstrap Aggregating на реальних даних
- Дослідити фінансовий часовий ряд компанії Yahoo! Finance

Критерій знаходження оптимальних моделей

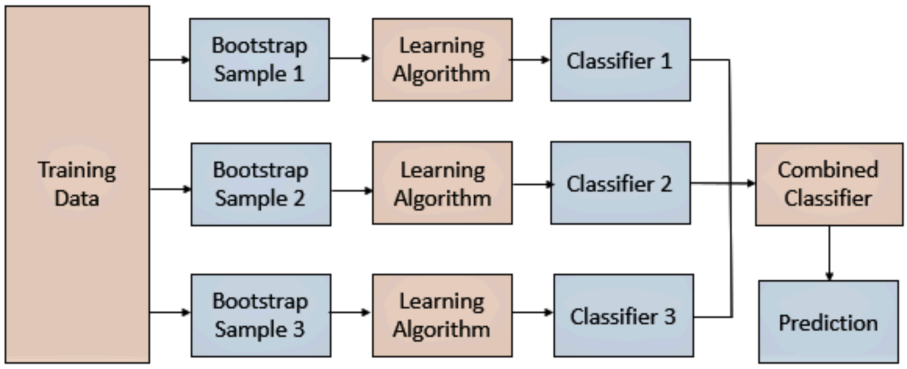
- Оскільки всі 3 моделі вважаються оптимальними для фінансового часового ряду такої природи як у дипломній роботі, критерієм вибору найкращої буде значення середньоквадратичної помилки (MSE)

$$\mathbf{L} = \frac{1}{\mathbf{N}} [\, \Sigma(\hat{\mathrm{Y}} - \mathrm{Y})^2]$$

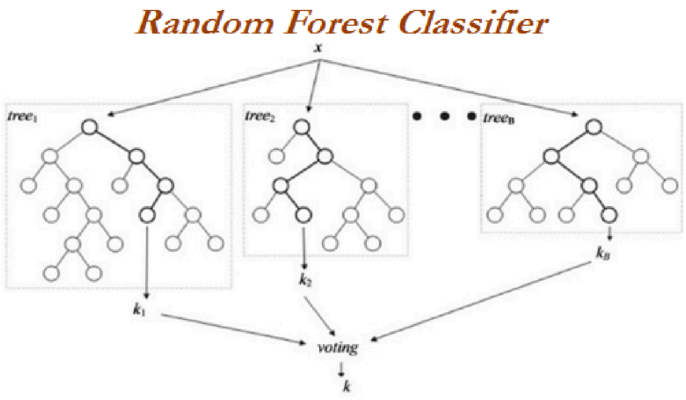
Моделі прогнозу

- Будуть представлені три моделі
- AdaBoost – адаптивний бустинг
- Random Forest – модель, котра вважається оптимальною для вибірки Yahoo Finance
- Bagging – покращення, котре дає ще меншу помилку при мінімальних додаткових обчислювальних ресурсах та часу на навчання

Bagging



Random Forest



AdaBoost (Будуватиметься ансамбль класифікаторів.

Формально, роботу АдаБуст можно записати так:

Дано: X^l - обучаюча вибірка;

$b_1,\cdots,b_T$ - базові алгоритми класифікації;

1. Ініціалізуємо ваги елементів:

$$p_i = 1/l,\ i = 1,\cdots,l;$$

2. Для будь-якого $t = 1,\cdots,T$, доки не виконується критерій зупинки.

2.1 Знаходимо класифікатор

$b_t : X \to \{-1,+1\}$, такий, що $b_t = \arg\min_t Q(b, W^l)$;

2.2 Змінюємо коефієнт взваженого голосування для b_t:

$$\alpha_t = \frac{1}{2}\ln\frac{1-Q(b,W^l)}{Q(b,W^l)};$$

2.3 Обчислюємо вагу елементів заново:

$$w_i = w_i \exp(-\alpha_t y_i b_t(x_i)),\ i = 1,\cdots,l;$$

2.4 Нормуємо їх:

$$w_0 = \sum_{j=1}^{l} w_j;\ w_i = w_i / w_0,\ i = 1,\cdots,l;$$

3. Остаточно повертаємо:

$$a(x) = sign\left(\sum_{i=1}^{T} \alpha_i b_i(x)\right)$$

Часовий фінансовий ряд Yahoo! Finance

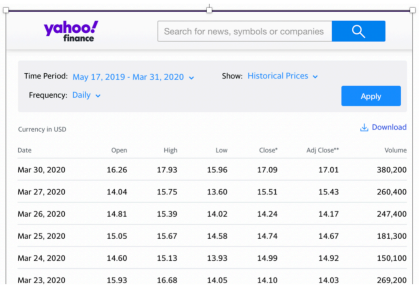
- Вищевказані три алгоритми будуть застосовані для прогнозування щоденних прибутків для запасів компанії, використовуючи попередні дані за три дні повернення вкладень.
- **Yahoo! Finance** представляє інформацію о фінансах компанії Yahoo! з 1998 року. Являє собою головного поставника такої інформації у США зі стамільйоною аудиторією станом на 2020 рік. Дані представляють собою інформацію про рейтинги акцій та фінансових звітів компанії.

Особливості даних

- динамічно змінюється з часом та є можливість оновлювати моделі кожен день за необхідністю постійно додаючи нові об’єкти
- дані повністю покривають усі слабкі моменти кожної моделі
- Це складне завдання не в останню чергу через те, що дані, мають низьке співвідношення сигнал / шум, а й тому, що такі дані послідовно корельовані. Це означає, що обрані вибірки не є справді незалежними одна від одної.

Наразі буде застосовано стандартний розрив навчальних тестувань, оскільки в дипломній роботі акцент робиться на порівнянні помилок між моделями, а не на абсолютній помилці, досягнутій для кожної.

Вигляд датасету



Date	Volume	Today	Lag1	Lag2	Lag3
2010-01-08	9830500	2.707696	-1.701323	-1.811569	0.589999
2010-01-11	8779400	-2.404139	2.707696	-1.701323	-1.811569
2010-01-12	9096300	-2.271506	-2.404139	2.707696	-1.701323
2010-01-13	10723200	1.382020	-2.271506	-2.404139	2.707696
2010-01-14	9774900	-1.363180	1.382020	-2.271506	-2.404139
...	...	...	...	...	...
2020-03-25	6479100	-2.796764	1.958663	3.073522	-1.852280
2020-03-26	6221300	3.693316	-2.796764	1.958663	3.073522
2020-03-27	5387900	-2.832539	3.693316	-2.796764	1.958663
2020-03-30	6126100	3.360348	-2.832539	3.693316	-2.796764
2020-03-31	5123600	-0.724559	3.360348	-2.832539	3.693316

Опис датасету

- Date – точна дата, коли були зафіксовані значення
- Volume – об’єм прибутку компанії за цей день
- Today – повернення вкладень компанії за цей день
- Lag1 - повернення вкладень компанії за попередній день
- Lag2 - повернення вкладень компанії за два дні до дати у стовпчику Date
- Lag3 - повернення вкладень компанії за три дні до дати у стовпчику Date

Таким чином, датасет складатиметься з матриці 2574 (кількість днів) на 5 (кількість характеристик), тобто з 12870 об’єктів.

Пакети Python

- Scikit-learn надає ряд алгоритмів навчання з вчителем та без через узгоджений інтерфейс в Python.
- 1)NumPy: базовий пакет n-мірного масиву
- 2)SciPy: фундаментальна бібліотека для наукових обчислень
- 3)Matplotlib: Комплексна 2D / 3D графіка
- 4)Sympy: Символічна математика
- 5)Pandas: структури даних і аналіз

Реалізація

In [9]:
```
if __name__ == "__main__":
    random_state = 42
    n_jobs = 4
    n_estimators = 1000
    step_factor = 10
    axis_step = int(n_estimators/step_factor)
```

Спільні параметри

```
start = datetime.datetime(2010, 1, 1)
end = datetime.datetime(2020, 3, 31)
amzn = create_lagged_series("AMZN", start, end, lags=3)
amzn.dropna(inplace=True)
```

Перетворення датасету

In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=random_state)
```

Розділені вибірки на навчальну
на тестувальну

In [23]:
```
estimators = np.zeros(axis_step)
bagging_mse = np.zeros(axis_step)
rf_mse = np.zeros(axis_step)
boosting_mse = np.zeros(axis_step)
```

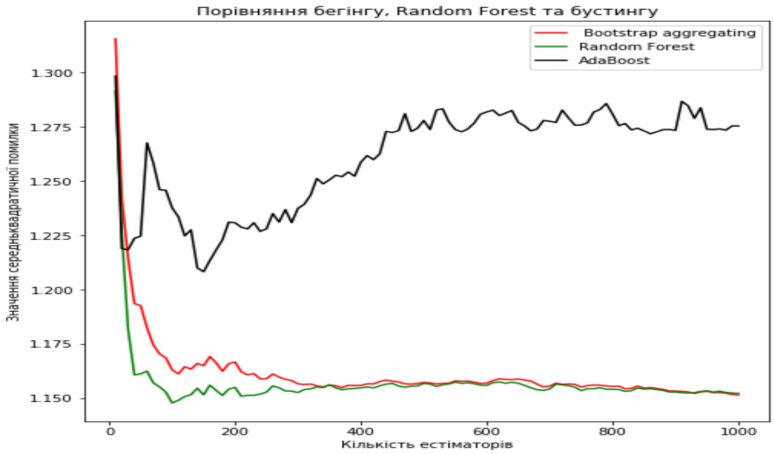
Початкові значення для моделей

Приклад реалізації та роботи Bagging

In [*]:
```
for i in range(0, axis_step):
    print("Bagging Estimator: %d of %d..." % (
        step_factor*(i+1), n_estimators)
    )
    bagging = BaggingRegressor(
        DecisionTreeRegressor(),
        n_estimators=step_factor*(i+1),
        n_jobs=n_jobs,
        random_state=random_state
    )
    bagging.fit(X_train, y_train)
    mse = mean_squared_error(y_test, bagging.predict(X_test))
    estimators[i] = step_factor*(i+1)
    bagging_mse[i] = mse
```

```
Bagging Estimator: 10 of 1000...
Bagging Estimator: 20 of 1000...
Bagging Estimator: 30 of 1000...
Bagging Estimator: 40 of 1000...
Bagging Estimator: 50 of 1000...
Bagging Estimator: 60 of 1000...
Bagging Estimator: 70 of 1000...
Bagging Estimator: 80 of 1000...
Bagging Estimator: 90 of 1000...
Bagging Estimator: 100 of 1000...
Bagging Estimator: 110 of 1000...
Bagging Estimator: 120 of 1000...
Bagging Estimator: 130 of 1000...
Bagging Estimator: 140 of 1000...
Bagging Estimator: 150 of 1000...
Bagging Estimator: 160 of 1000...
Bagging Estimator: 170 of 1000...
Bagging Estimator: 180 of 1000...
Bagging Estimator: 190 of 1000...
Bagging Estimator: 200 of 1000...
Bagging Estimator: 210 of 1000...
```

Порівняння результатів роботи



Висновки

- Побудовано модель бустингу AdaBoost з непоганим значенням помилки, котра не перенавчилась на вибірці.
- Досягнено кращого за Random Forest результату, котрий вважається оптимальним алгоритмом для ряду такої природи як у Yahoo Finance
- Кращий результат вдалось досягти моделі з використанням підходу bootstrap aggregating
- Досліджено фінансовий часовий ряд компанії Yahoo! Finance

ДОДАТОК Б ПРОГРАМНИЙ ПРОДУКТ НАВЧАННЯ

```
import datetime

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import seaborn as sns
import sklearn
from sklearn.ensemble import (
    BaggingRegressor, RandomForestRegressor, AdaBoostRegressor
)
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.tree import DecisionTreeRegressor

def create_lagged_series(symbol, start_date, end_date, lags=3):

    ts = web.DataReader(
        symbol, "yahoo", start_date, end_date
    )



    tslag = pd.DataFrame(index=ts.index)
    tslag["Today"] = ts["Adj Close"]
    tslag["Volume"] = ts["Volume"]
```

```
    for i in range(0,lags):
        tslag["Lag%s" % str(i+1)] = ts["Adj Close"].shift(i+1)


    tsret = pd.DataFrame(index=tslag.index)
    tsret["Volume"] = tslag["Volume"]
    tsret["Today"] = tslag["Today"].pct_change()*100.0

    for i in range(0,lags):
        tsret["Lag%s" % str(i+1)] = tslag[
            "Lag%s" % str(i+1)
        ].pct_change()*100.0
    tsret = tsret[tsret.index >= start_date]
    return tsret

if __name__ == "__main__":
    random_state = 42
    n_jobs = 16
    n_estimators = 1000
    step_factor = 10
    axis_step = int(n_estimators/step_factor)

    start = datetime.datetime(2000, 1, 1)
    end = datetime.datetime(2020, 3, 31)
    amzn = create_lagged_series("AMZN", start, end, lags=3)
    amzn.dropna(inplace=True)

    X = amzn[["Lag1", "Lag2", "Lag3"]]
```

```
    y = amzn["Today"]
    X = scale(X)
    y = scale(y)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=random_state)
    estimators = np.zeros(axis_step)
    bagging_mse = np.zeros(axis_step)
    rf_mse = np.zeros(axis_step)
    boosting_mse = np.zeros(axis_step)
for i in range(0, axis_step):
        print("Bagging Estimator: %d of %d..." % (
            step_factor*(i+1), n_estimators)
        )
        bagging = BaggingRegressor(
            DecisionTreeRegressor(),
            n_estimators=step_factor*(i+1),
            n_jobs=n_jobs,
            random_state=random_state
        )
        bagging.fit(X_train, y_train)
        mse = mean_squared_error(y_test, bagging.predict(X_test))
        estimators[i] = step_factor*(i+1)
        bagging_mse[i] = mse

    for i in range(0, axis_step):
        print("Random Forest Estimator: %d of %d..." % (
            step_factor*(i+1), n_estimators)
        )
        rf = RandomForestRegressor(
```

```
            n_estimators=step_factor*(i+1),
            n_jobs=n_jobs,
            random_state=random_state
        )
        rf.fit(X_train, y_train)
        mse = mean_squared_error(y_test, rf.predict(X_test))
        estimators[i] = step_factor*(i+1)
        rf_mse[i] = mse
for i in range(0, axis_step):
        print("Boosting Estimator: %d of %d..." % (
            step_factor*(i+1), n_estimators)
        )
        boosting = AdaBoostRegressor(
            DecisionTreeRegressor(),
            n_estimators=step_factor*(i+1),
            random_state=random_state,
            learning_rate=0.01
        )
        boosting.fit(X_train, y_train)
        mse = mean_squared_error(y_test, boosting.predict(X_test))
        estimators[i] = step_factor*(i+1)
        boosting_mse[i] = mse
    plt.figure(figsize=(8, 8))
    plt.title('Порівняння бегінгу, Random Forest та бустингу')
    plt.plot(estimators, bagging_mse, 'b-', color="red", label=' Bootstrap
aggregating')
    plt.plot(estimators, rf_mse, 'b-', color="green", label='Random Forest')
    plt.plot(estimators, boosting_mse, 'b-', color="black", label='AdaBoost')
    plt.legend(loc='upper right')
```

```
plt.xlabel('Кількість естіматорів')
plt.ylabel('Значення середньквадратичної помилки')
plt.show()
```